ソフトウェア工学 64－8
（1989. 2. 2）

仕様記述構成の形式化

南祐昭　　米崎直樹

東京工業大学工学部情報工学科

概要

仕様記述において仕様が構成される過程を明らかにし、その過程の形式化を行う。既存の仕様から新しい仕様を構成する際には通常、一般化、特殊化の操作を行なうが、まず、それらの操作を分類整理し、仕様構成規則として形式化する。次に、こうして得られた構成方法をオペレータとして持ち、仕様記述言語をオブジェクト言語とするメタレベルの言語を設計し、その言語に対してそれぞれの構成規則に対応した意味を定め、推論規則の体系を構成する。本研究では、オブジェクト言語として、ある形式の一階の述語論理式を考え、メタ言語の意味として解の存在証明によって導出される関数を与える。したがって、既存の仕様から新たな仕様を構成した場合、その仕様の存在証明を構成する方法が与えられる。

Formalization of Specification Construction

Hiroaki Minami　　Naoki Yonezaki

Department of Computer Science,
Tokyo Institute of Technology,
O-okayama, Meguro-ku, Tokyo, 152, Japan

Abstract

The formalization of the process of constructing a specification is presented. If a specification is obtained by modifying another specification or combining the other specifications, we generally use the two kinds of operations, i.e. specialization and generalization. These operations are formally defined as specification construction rules syntactically. Then, we introduce the meta-level language which has the construction rules as operators and the specification language as an object-level language. The semantics of the meta-language is defined by the justification rules of the existential proof of the values which satisfy a specification. This rule can be also considered as the construction rule of a functional program which satisfies the specification.

仕様構成式では、parameter instantiation 規則を用いてスキーマ indsch からこの仕様定義を生成する。

$$Plus = \text{param-inst}(\text{indsch}, \ldots)$$

意味規則を用いて解の存在証明を行う場合、まず帰納法で使用する変数を選び、仕様で定義されたモジュールの入出力を決める。ここでは x, y を入力 z を出力として選び x について帰納法を行う。param-inst 規則の意味規則を用いれば、

$$\frac{\Gamma \rhd \forall x, y \exists z \text{base}(x, Plus) : f_B \quad \Gamma \rhd \forall x, y \exists z \text{ind}(x, Plus) : f_I}{\Gamma \rhd \forall x, y \exists z Plus : f_B \text{ else } f_I \text{ else } \bot}$$

base case : $\forall x, y \exists z \text{base}(x, Plus)$に対応して証明される式は $x = 0$ のもとで

$$\forall y \exists z (y = z)$$

である。これより x, y を入力とし z を出力とする導出項 $f_B(x, y) = z$ は、$x = 0$ のもとで $f_B(x, y) = y$、すなわち、

$$f_B(x, y) = \textbf{if } x = 0 \textbf{ then } y$$

となる。

induction step : $\forall x', y \exists z Plus(x', y, z)$の解の存在証明がありその導出項を $f(x', y) = z$ と仮定する。ここで、$x = s(x')$のときの $\forall x, y \exists z Plus(x, y, z)$ の解の存在証明を求める。$\forall x, y \exists z \text{ind}(x, Plus)$に対応して証明される式は $x = s(x')$のもとで

$$\forall x', y \exists z, z' (Plus(x', y, z') \wedge z = s(z')))$$

である。帰納法の仮定より $Plus(x', y, z')$は $\forall x', y \exists z'$のときの解の存在証明があり導出項は $f(x', y) = z'$である。これと $z = s(z')$より $z = s(f(x', y))$。従って導出項 $f_I(x, y) = z$は $x = s(x')$のもとで $f_I(x, y) = s(f(x', y))$となる。すなわち、

$$f_I(x, y) = \textbf{if } x = s(x') \textbf{ then } s(f(x', y))$$

となる。

最終的に得られる導出項 fは次のようになる。

$$\begin{aligned} f(x, y) \quad = \quad & \textbf{if } x = 0 \textbf{ then } y \\ & \textbf{else if } x = s(x') \textbf{ then } s(f(x', y)) \\ & \textbf{else } \bot \end{aligned}$$

5 まとめ

一般化と特殊化という概念に基づいて仕様の構成規則を形式化した。このような形式化は仕様記述のスタイルにある型をはめることになり、それが我々が頭の中で概念を構成する方法に対応しているならば、自然で効率のいい記述を可能とするはずである。また、構成規則の意味としては仕様に対応する関数型プログラムの意味を与えたが、これは半自動的なプログラムの合成に有用に用いられるであろう。今後、形式化された規則を計算機に実装することにより、より現実的な具体例を計算機の援助のもとで記述できるようにすることが必要である。

本稿で示したような形式化を行い実際の問題と比較していくことがソフトウェアエンジニアリングの立場からは重要であろう。

References

[1] 榎本 肇、米崎直樹、佐伯元司. Tell/nsl における語句定義構造と意味規則. , 第 *30* 回情報処理学会全国大会論文集, 1985 .

[2] R. L. Constable et al. In *Implementing Mathematics with the Nuprl Proof Development System*, 1986.

仕様Γの要素は有限個であり、各仕様定義 A_i の定義本体に現れる仕様定義も有限個なので、$R^+(A), R^*(A)$ はある有限集合に定義される。

仕様定義 A が再帰的定義を含んでいる場合、次の関係が成り立つ。

$$A \in R^+(A)$$

3章以降で扱う仕様は定義の完全性および無矛盾性を満たした仕様である。仕様の定義の完全性および無矛盾性についての定義を次に示す。

定義 2.3 (仕様定義の完全性) 仕様Γ中のどのような仕様定義 A_i に対しても、A_i の入出力 $x_{i1}, \ldots, x_{in}$ について、

$$\Gamma \vdash A_i(x_{i1}, \ldots, x_{in})$$

または

$$\Gamma \vdash \neg A_i(x_{i1}, \ldots, x_{in})$$

となるとき、仕様Γの定義は完全である。

例 2.1 2つの仕様定義 A, C

$$\begin{aligned} A &\leftrightarrow \mathcal{F}_A(B, C) \\ C &\leftrightarrow \mathcal{F}_C(C) \end{aligned}$$

で構成される次のような仕様Γ

$$\Gamma = \{A, C\}$$

は、Γ中に未定義の述語 B を使用している仕様定義 A を含んでいるので不完全な仕様である。

定義 2.4 (仕様の無矛盾性) 仕様Γ中の仕様定義 $A_1, \ldots, A_n$ が条件

$$A_1 \wedge \ldots \wedge A_n \not\vdash \mathbf{false}$$

を満たしているとき、仕様Γは無矛盾である。

3 仕様構成規則

既存の仕様から新しい仕様を作り出す際に、両者の間に存在する関係に注目すると、いくつかの種類に分類できることがわかる。これを規則として形式化したものが仕様構成規則である。

仕様構成規則を選び出す際の基本的な方針は以下の通りである。

- 仕様記述言語である一階述語論理式の構文則も構成規則の一つと考えることができるが、より人間の概念構成の直観に近い規則となるようにする。
- しかし、複雑な構成規則は導入せず、構成方法はできるだけ簡単にし、一般化・特殊化として捉えられるものとする。

現在、形式化された仕様構成規則は9種類ある。仕様構成規則は仕様を特殊化する規則(specialize rule)と一般化する規則(generalize rule)に大別され、それらは通常対で存在する。以下に仕様構成規則とその操作を示す。

1. ∧-composition 規則 [specialization rule]

   既存の仕様定義 $A_1, \ldots, A_n (n \geq 2)$ を論理積で結合して新しい仕様定義 A' を作り出す。

   $$\begin{aligned} & A_1(x_{11}, \ldots, x_{1k_1}) \leftrightarrow \mathcal{F}_1 \\ & \quad \vdots \\ & A_n(x_{n1}, \ldots, x_{nk_n}) \leftrightarrow \mathcal{F}_n \\ & \quad \Downarrow \quad \wedge\text{-comp} \\ & A'(x'_1, \ldots, x'_{k'}) \leftrightarrow \exists x'_{k'+1} \ldots x'_l \\ & \qquad ((\mathcal{F}_1 \wedge \ldots \wedge \mathcal{F}_n) \cdot \{x'_1/V_1, \ldots, x'_l/V_l\}) \end{aligned}$$

   存在記号で束縛される変数はなくてもよい。また、$\cdot\{\ldots, x_i/V_i, \ldots\}$ は、変数の集合 V_i の要素の出現を変数 x_i で置換する操作を表す。V_i は既存の仕様定義中に出現するすべての自由変数の部分集合で、次の条件を満たす。

   $$\begin{aligned} V_i \bigcap V_j &= \emptyset \qquad (i \neq j) \\ \bigcup_{1 \leq i \leq l} V_i &= \bigcup_{1 \leq j \leq n} FV(\mathcal{F}_j) \\ x_{p_1 q_1}, x_{p_2 q_2} \in V_i &\rightarrow p_1 \neq p_2 \end{aligned}$$

2. ∧-decomposition 規則 [generalization rule]

   n 個の論理積で結合された式より構成される既存の仕様定義 A からその構成式の一部をなす仕様定義 $A_m (1 \leq m \leq n)$ を取り除き新しい仕様定義 A' とする。これは∧-composition 規則と対の規則である。

   $$\begin{aligned} & A(x_1, \ldots, x_k) \leftrightarrow \exists y_1 \ldots y_l (\mathcal{F}_1 \wedge \ldots \wedge \mathcal{F}_n) \\ & A_m(x_{m1}, \ldots, x_{mk_m}) \leftrightarrow \mathcal{F}_m \\ & \quad \Downarrow \quad \wedge\text{-decomp} \\ & A'(x'_1, \ldots, x'_{k'}) \leftrightarrow \exists y'_1 \ldots y'_{l'} \\ & \qquad (\mathcal{F}_1 \wedge \ldots \wedge \mathcal{F}_{m-1} \wedge \mathcal{F}_{m+1} \wedge \ldots \wedge \mathcal{F}_n) \end{aligned}$$

A'に含まれる自由変数x'_iは、A中の自由変数からA_mにだけ含まれる自由変数を除いたものである。すなわち、

$$FV(A') = \bigcup_{1\leq i\leq n, i\neq m} FV(\mathcal{F}_i)$$

A'の定義本体中で束縛される変数$y'_1,\ldots,y'_{l'}$についても同様で、$y_1,\ldots,y_l$のうち、$\mathcal{F}_m$にだけ現れる変数を除いたものである。

3. **∨-composition 規則** [genaralization rule]

既存の仕様定義$A_1,\ldots,A_n(n\geq 2)$を論理和で結合して新しい仕様定義A'を作り出す。

$$A_1(x_{11},\ldots,x_{1k_1}) \leftrightarrow \mathcal{F}_1$$
$$\vdots$$
$$A_n(x_{n1},\ldots,x_{nk_n}) \leftrightarrow \mathcal{F}_n$$
$$\Downarrow \quad \vee\text{-comp}$$
$$A'(x'_1,\ldots,x'_l) \leftrightarrow (\mathcal{F}_1 \vee \ldots \vee \mathcal{F}_n)\cdot\{x'_1/V_1,\ldots,x'_l/V_l\}$$

変数の集合V_iの満たす条件は∧-composition 規則と同じである。

4. **∨-decomposition 規則** [specialization rule]

n 個の論理和で結合された式より構成される既存の仕様定義Aからその構成式の一部をなす仕様定義$A_m(1\leq m\leq n)$を取り除き新しい仕様定義A'とする。これは、∨-composition 規則と対の規則である。

$$A(x_1,\ldots,x_k) \leftrightarrow \mathcal{F}_1 \vee \ldots \vee \mathcal{F}_n$$
$$A_m(x_{m1},\ldots,x_{mk_m}) \leftrightarrow \mathcal{F}_m$$
$$\Downarrow \quad \vee\text{-decomp}$$
$$A'(x'_1,\ldots,x'_{k'}) \leftrightarrow \mathcal{F}_1 \vee \ldots \vee \mathcal{F}_{m-1} \vee \mathcal{F}_{m+1} \vee \ldots \vee \mathcal{F}_n$$

A'の変数x'_iの満たす条件は∧-decomposition 規則と同じである。

5. **→-composition 規則** [generalization rule]

既存の仕様定義$A_0,A_1,\ldots,A_n(n\geq 2)$を含意で結合して新しい仕様定義$A'$を作り出す。

$$A_0(x_{01},\ldots,x_{0k_0}) \leftrightarrow \mathcal{F}_0$$
$$A_1(x_{11},\ldots,x_{1k_1}) \leftrightarrow \mathcal{F}_1$$
$$\vdots$$
$$A_n(x_{n1},\ldots,x_{nk_n}) \leftrightarrow \mathcal{F}_n$$
$$\Downarrow \quad \rightarrow\text{-comp}$$
$$A'(x'_1,\ldots,x'_{k'}) \leftrightarrow \forall x'_{k'+1}\ldots x'_l (((\mathcal{F}_n \wedge \cdots \wedge \mathcal{F}_1) \rightarrow \mathcal{F}_0)\cdot\{x'_1/V_1,\ldots,x'_l/V_l\})$$

変数の集合V_iの満たす条件は∧-composition 規則と同じである。

6. **→-decomposition 規則** [specialization rule]

n 個の含意で結合された式より構成される既存の仕様定義Aからその構成式の一部をなす仕様定義$A_m(1\leq m\leq n)$を取り除き新しい仕様定義A'とする。この規則は、→-composition 規則と対の規則である。

$$A(x_1,\ldots,x_k) \leftrightarrow \forall y_1\ldots y_l((\mathcal{F}_n \wedge \ldots \wedge \mathcal{F}_1) \rightarrow \mathcal{F}_0)$$
$$A_m(x_{m1},\ldots,x_{mk_m}) \leftrightarrow \mathcal{F}_m$$
$$\Downarrow \quad \rightarrow\text{-decomp}$$
$$A'(x'_1,\ldots,x'_{k'}) \leftrightarrow \forall y'_1\ldots y'_{l'} ((\mathcal{F}_n \wedge \ldots \wedge \mathcal{F}_{m+1} \wedge \mathcal{F}_{m-1} \wedge \ldots \wedge \mathcal{F}_1) \rightarrow \mathcal{F}_0)$$

A'の変数x'_iの満たす条件は∧-decomposition 規則と同じである。

7. **instantiation 規則** [specialization rule]

既存の仕様定義A中の自由変数x_mを項tで置換し新しい仕様定義A'とする。t中の変数$x_{m1},\ldots,x_{mk}$は代入することにより新たに束縛されることはない。

$$A(x_1,\ldots,x_n) \leftrightarrow \mathcal{F}$$
$$\Downarrow \quad \text{inst}$$
$$A'(x_1,\ldots,x_{m-1},x_{m+1},\ldots,x_n,x_{m1},\ldots,x_{mk}) \leftrightarrow \mathcal{F}[t/x_m]$$

$\mathcal{F}[t/x_m]$は$\mathcal{F}$中の変数x_mを項tで置換する操作を表す。

8. **parameterization 規則** [generalization rule]

既存の仕様定義A中の項tを自由変数x_mで置換し新しい仕様定義A'とする。項tは定数あるいは自由変数$x_{m1},\ldots,x_{mk}\in FV(A)$を含んだ項で

ある。この規則は、instantiation 規則と対の規則である。

$$A(x_1,\ldots,x_n) \leftrightarrow \mathcal{F}$$
$$\Downarrow \quad \text{param}$$
$$A'(x_1,\ldots,x_{n'},x_m) \leftrightarrow \mathcal{F}[x_m/t]$$

ただし、A'中の自由変数 $x_1,\ldots,x_{n'}$ は A 中の自由変数から項 t 中にだけ現れる自由変数を除いたものである。

次の規則は、仕様から仕様を生成する規則ではなく、スキーマから仕様を生成する規則である。この規則は型あるいは仕様を帰納的に定義する際に使用する。

9. **parameter instantiation** 規則

使用するスキーマを S、スキーマに引数として与える述語を P_i、関数を f_j、定数を c_k およびそれらの引数の数 n_P, n_f, n_c とするとき、新しい仕様 P_0 を生成する操作は次のように表される。

$$\pi P_0.\nu n_P.\nu n_f.\nu n_c.\pi P_1.\ldots.\pi P_{n_P}.$$
$$\gamma f_1.\ldots.\gamma f_{n_f}.\sigma c_1.\ldots.\sigma c_{n_c}.S$$
$$\Downarrow \quad \text{param-inst}$$
$$P_0(x_1,\ldots,x_n) \leftrightarrow \mathcal{F}(x_1,\ldots,x_n)$$

スキーマはあらかじめ次の 2 つが用意されている。x, x'が帰納法で使用する変数である。

(a) 型仕様を生成するスキーマ(tindsch)

$$\pi P_0.\nu n_P.\pi P_1.\ldots.\pi P_{n_P}.\gamma f.\sigma c.$$
$$\{P_0(x) \leftrightarrow x = c \vee$$
$$\exists x', y_1,\ldots,y_{n_P}(x = f(x', y_1,\ldots,y_{n_P})$$
$$\wedge P_0(x') \wedge P_1(y_1) \wedge \ldots \wedge P_{n_P}(y_{n_P}))\}$$

(b) 一般の仕様を生成するスキーマ(indsch)

$$\pi P_0.\nu n_P.\xi\mathcal{F}_B.\xi\mathcal{F}_I.\gamma f.\sigma c.$$
$$\{P_0(x, y_1,\ldots,y_{n_P}) \leftrightarrow (x = c \wedge \mathcal{F}_B) \vee$$
$$\exists x', y'_1,\ldots,y'_{n_P}(x = f(x')$$
$$\wedge P_0(x', y'_1,\ldots,y'_{n_P}) \wedge \mathcal{F}_I)\}$$

$\mathcal{F}_B, \mathcal{F}_I$ は変数 y, y' の関係を表す述語である。

4 仕様構成言語

本章では仕様構成言語の構文と意味を定義し、さらにその推論規則を定める。仕様構成言語は、2 章で述べた仕様記述言語をオブジェクトレベルの言語とし、3 章で述べた仕様構成規則をオペレータとして持つメタレベルの言語である。

4.1 構文

まず、仕様構成規則を用いて得られる仕様を値として持つ仕様構成式の定義を行う。

定義 4.1 (仕様構成式) 次の条件のいずれかを満たす式のみが仕様構成式である。

1. parameter instantiation 規則によりスキーマから生成した仕様定義式または仕様名は仕様構成式である。
2. 仕様構成式に仕様構成規則を施して得られた式は仕様構成式である。

一般に仕様構成式 $\Psi_1,\ldots,\Psi_n$ からオペレーション op を用いて仕様構成式 Ψ を生成する操作は次のように表される。

$$\Psi = \mathrm{op}(\Psi_1,\ldots,\Psi_n, params\ldots)$$

$params\ldots$ は必要に応じて存在し、それはオペレーションにより異なる。各規則ごとの構文は付録に示す。付録の表記中、$param$ の部分で使われる変数の用法は 3 章で示した規則ごとの操作の中に出てくる変数の用法に等しい。以下に仕様構成式による仕様構成の表記例を示す。

例 4.1

$Brother(x,y)$	xとyは兄弟
$Father(z,w)$	zはwの父

上の 2 つの定義から定義 $Uncle$ を作ると次のようになる。

$$Uncle(x,w) \leftrightarrow \exists v(Brother(x,v) \wedge Father(v,w))$$

仕様構成式ではこの操作を∧-composition 規則を用いて、

$$Uncle =$$
$$\wedge\text{-comp}(Brother, Father, \{\langle x\rangle, \langle w\rangle\}, \{\langle y,z\rangle\})$$

と表す。引数 $\{\langle x\rangle, \langle w\rangle\}$ は仕様 $Uncle$ 中に残る自由変数を表す。また、引数 $\{\langle y,z\rangle\}$ は、元の仕様の自由変数 y, z を同じ変数に置換しヨで束縛することを意味する。

例 4.2 乗算の仕様定義

$$Times(x, y, z) \qquad (z = x \times y)$$

から instantiation 規則を用いて 3 の倍数の定義 $Mult3(x, z)$を作る場合、変数 yを定数 3 で置換すればよい。この仕様は次のように表される。

$$Mult3 = \mathrm{inst}(Times, y, 3)$$

例 4.3 自然数の配列の型定義 $Array(x)$を生成する場合、スキーマ tindsch にパラメータ

$$\begin{aligned} P_0 &= Array \\ n_P &= 2 \\ P_1 &= Index \\ P_2 &= N \\ f &= set \\ c &= \mathbf{NIL} \end{aligned}$$

を代入すれば、次の型定義式が生成される。

$$\begin{aligned} &Array(x) \leftrightarrow \\ &\quad x = \mathbf{NIL} \vee \\ &\qquad \exists x', y_1, y_2 (x = set(x', y_1, y_2) \\ &\qquad \wedge Array(x') \wedge Index(y_1) \wedge N(y_2)) \end{aligned}$$

仕様構成式ではこの操作を parameter-instantiation 規則を使って次のように表す。

$$\begin{aligned} &Array = \\ &\quad \text{param-inst}(S_1, \\ &\qquad \langle Array, 2, Index, N, set, \mathbf{NIL} \rangle) \end{aligned}$$

4.2 意味

ここでは、仕様構成式の意味を定義し、またそれぞれの仕様構成式に対する意味規則を示す。さらにこの意味規則を用いた具体的な存在証明の例について述べる。

定義 4.2 (仕様構成式の評価) 仕様構成式Ψを評価して得られる仕様定義式を $value(\Psi)$で表す。また、このときの仕様定義式の定義頭部を A_Ψ、定義本体を $\mathcal{F}_\Psi$ で表す。すなわち、

$$value(\Psi) = \forall \ldots x_i \ldots (A_\Psi \leftrightarrow \mathcal{F}_\Psi)$$

定義 4.3 (仕様構成式の意味) 仕様Γの下での仕様構成式Ψの意味Ωを次のように表記する。

$$\Gamma \rhd \forall X_\alpha \exists X_\beta \Psi : \Omega$$

X_α, X_βは仕様定義式 $value(\Psi)$の最も外側で全称的に束縛される変数の列である。意味Ωは、値の存在証明

$$\Gamma \vdash \forall X_\alpha \exists X_\beta A_\Psi$$

すなわちΓを仮定して$\forall X_\alpha \exists X_\beta A_\Psi$を構成的に証明したときに得られる導出関数とする。

$$\Gamma \not\vdash \forall X_\alpha \exists X_\beta A_\Psi$$

のとき、Ωは未定義($\perp$)となる。すなわち、

$$\Gamma \rhd \forall X_\alpha \exists X_\beta \Psi : \perp$$

$\Gamma \subset R^+(A_\Psi)$のときも$\Omega$は$\perp$である。

ここで、任意の仕様構成式の意味は以下のように仕様構成式に従って帰納的に定義される。

4.3 意味規則

各仕様構成規則に対応する意味規則を付録に示す。規則中に現れる$\forall X_\alpha$、$\exists X_\beta$はそれぞれ全称的に束縛される変数の列、存在的に束縛される変数の列を表し、仕様構成式の定義するモジュールの入出力に対応する。仕様構成式中の自由変数は必ず全称か存在のいずれかに対応しなければならない。すなわち、仕様構成式Ψに対応する仕様定義式を A_Ψとするとき以下の条件を満たす。

$$\begin{aligned} X_\alpha \cup X_\beta &= FV(A_\Psi) \\ X_\alpha \cap X_\beta &= \emptyset \end{aligned}$$

また、推論規則中の仮言部と結言部に現れる対応づけられる変数は同じ限定子が付けられる。

4.4 証明例

ここでは、意味規則を用いて加算の仕様の解の存在証明を行う例を示す。加算の仕様定義 *Plus* は仕様記述言語では以下のように帰納的に定義される。

$$\begin{aligned} &Plus(x, y, z) \leftrightarrow (x = 0 \wedge y = z) \vee \\ &\quad \exists x', z' (x = s(x') \wedge Plus(x', y, z') \wedge z = s(z')) \end{aligned}$$

仕様構成式では、parameter instantiation 規則を用いてスキーマ indsch からこの仕様定義を生成する。

$$Plus = \text{param-inst}(\text{indsch}, \ldots)$$

意味規則を用いて解の存在証明を行う場合、まず帰納法で使用する変数を選び、仕様で定義されたモジュールの入出力を決める。ここでは x, y を入力 z を出力として選び x について帰納法を行う。param-inst 規則の意味規則を用いれば、

$$\frac{\Gamma \rhd \forall x, y \exists z \text{base}(x, Plus) : f_B \quad \Gamma \rhd \forall x, y \exists z \text{ind}(x, Plus) : f_I}{\Gamma \rhd \forall x, y \exists z Plus : f_B \text{ else } f_I \text{ else } \bot}$$

base case : $\forall x, y \exists z \text{base}(x, Plus)$に対応して証明される式は $x = 0$ のもとで

$$\forall y \exists z (y = z)$$

である。これより x, y を入力とし z を出力とする導出項 $f_B(x, y) = z$ は、$x = 0$ のもとで $f_B(x, y) = y$、すなわち、

$$f_B(x, y) = \textbf{if } x = 0 \textbf{ then } y$$

となる。

induction step : $\forall x', y \exists z Plus(x', y, z)$の解の存在証明がありその導出項を $f(x', y) = z$ と仮定する。ここで、$x = s(x')$のときの $\forall x, y \exists z Plus(x, y, z)$ の解の存在証明を求める。$\forall x, y \exists z \text{ind}(x, Plus)$に対応して証明される式は $x = s(x')$のもとで

$$\forall x', y \exists z, z' (Plus(x', y, z') \wedge z = s(z')))$$

である。帰納法の仮定より $Plus(x', y, z')$は $\forall x', y \exists z'$のときの解の存在証明があり導出項は $f(x', y) = z'$である。これと $z = s(z')$より $z = s(f(x', y))$。従って導出項 $f_I(x, y) = z$は $x = s(x')$のもとで $f_I(x, y) = s(f(x', y))$となる。すなわち、

$$f_I(x, y) = \textbf{if } x = s(x') \textbf{ then } s(f(x', y))$$

となる。

最終的に得られる導出項 fは次のようになる。

$$\begin{aligned} f(x, y) \quad = \quad & \textbf{if } x = 0 \textbf{ then } y \\ & \textbf{else if } x = s(x') \textbf{ then } s(f(x', y)) \\ & \textbf{else } \bot \end{aligned}$$

5 まとめ

一般化と特殊化という概念に基づいて仕様の構成規則を形式化した。このような形式化は仕様記述のスタイルにある型をはめることになり、それが我々が頭の中で概念を構成する方法に対応しているならば、自然で効率のいい記述を可能とするはずである。また、構成規則の意味としては仕様に対応する関数型プログラムの意味を与えたが、これは半自動的なプログラムの合成に有用に用いられるであろう。今後、形式化された規則を計算機に実装することにより、より現実的な具体例を計算機の援助のもとで記述できるようにすることが必要である。

本稿で示したような形式化を行い実際の問題と比較していくことがソフトウェアエンジニアリングの立場からは重要であろう。

References

[1] 榎本 肇、米崎直樹、佐伯元司. Tell/nsl における語句定義構造と意味規則. , 第 *30* 回情報処理学会全国大会論文集, 1985 .

[2] R. L. Constable et al. In *Implementing Mathematics with the Nuprl Proof Development System*, 1986.

付録 仕様構成規則の構文と意味規則

1. ∧-composition 規則

$$\Psi' = \wedge\text{-comp}(\Psi_1, \ldots, \Psi_n, \{V_1, \ldots, V_k\}, \{V_{k+1}, \ldots, V_l\})$$

$$\frac{\Gamma_1 \rhd \forall X_{\alpha 1} \exists X_{\beta 1} \Psi_1 : f_1 \quad \ldots \quad \Gamma_n \rhd \forall X_{\alpha n} \exists X_{\beta n} \Psi_n : f_n}{\Gamma' \rhd \forall X_\alpha \exists X_\beta \Psi : f} \qquad (\text{ただし、} \Gamma = \bigcup_{1 \le i \le n} \Gamma_i)$$

全称限定変数 X_α(あるいは存在限定変数 X_β)に置換される変数の集合 $V_1, \ldots, V_k$ の要素は $f_i (1 \le i \le n)$ においても全称限定変数 $X_{\alpha i}$(あるいは存在限定変数 $X_{\beta i}$)でなければならない。$V_{k+1}, \ldots, V_l$ に含まれる変数は各 f_i においては存在限定変数 $X_{\beta i}$ に含まれる。X_α に含まれるすべての変数の値について各 f_i の $X_{\beta i}$ に対応する値がもし存在すれば等しいことが証明できれば、f の出力 $x_\beta \in X_\beta$ の値は x_β を出力として持つ f_i のそれに等しく、そうでなければ入力は $\perp$ である。これは関数のユニフィケーションにより得られる。

2. ∧-decomposition 規則

$$\Psi' = \wedge\text{-decomp}(\Psi, \Psi_m)$$

$$\frac{\Gamma \rhd \forall X_\alpha \exists X_\beta \Psi : f \qquad \Gamma_m \rhd \forall X_{\alpha m} \exists X_{\beta m} \Psi_m : f_m}{\Gamma \rhd \forall X_{\alpha'} \exists X_{\beta'} \Psi' : f'} \qquad (\text{ただし、} \Gamma_m \subseteq \Gamma)$$

f' は、f_m にのみ現れる入力と出力を f から除去した関数である。ただしここでは最大の関数が得られるとは保証されない。

3. ∨-composition 規則

$$\Psi' = \vee\text{-comp}(\Psi_1, \ldots, \Psi_n, \{V_1, \ldots, V_l\})$$

すべての $i (1 \le i \le n)$ について

$$\frac{\Gamma_i \rhd \forall X_{\alpha i} \exists X_{\beta i} \Psi_i : f_i}{\Gamma \rhd \forall X_\alpha \exists X_\beta \Psi' : \text{case}_i(f_i)} \qquad (\text{ただし、} \Gamma = \bigcup_{1 \le i \le n} \Gamma_i)$$

X_α, X_β と $X_{\alpha i}, X_{\beta i}$ の対応は∧-composition 規則と同じである。求める導出項 f_i は $f_1, \ldots, f_n$ のどれでもよいが、その際に選んだ導出項の位置の情報を case_i として加える。

4. ∨-decomposition 規則

$$\Psi' = \vee\text{-decomp}(\Psi, \Psi_m)$$

$$\frac{\Gamma \rhd \forall X_\alpha \exists X_\beta \Psi : \text{case}_i(f) \qquad \Gamma_m \rhd \forall X_{\alpha m} \exists X_{\beta m} \Psi_m : f_m}{\Gamma \rhd \forall X_{\alpha'} \exists X_{\beta'} \Psi' : f} \qquad (\text{ただし、} \Gamma_m \subseteq \Gamma, i \ne m)$$

$$\vdots$$

9. parameter instantiation 規則

$$\Psi' = \text{param-inst}(S, params \ldots)$$

S は仕様スキーマ、$params \ldots$ はスキーマに必要なパラメータを表す。

$$\frac{\Gamma \rhd \forall X_\alpha \exists X_\beta \text{base}(x, \Psi) : f_B \qquad \Gamma \rhd \forall X_\alpha \exists X_\beta \text{ind}(x, \Psi) : f_I}{\Gamma \rhd \forall X_\alpha \exists X_\beta \Psi' : f_B \text{ else } f_I \text{ else } \perp}$$

base(x, Ψ),ind(x, Ψ)はそれぞれ帰納法で使用する変数として x を選び仕様定義 A_Ψ を base case と induction step に分けたときの各部分である。