

代数的に仕様記述された言語の

プログラム検証

An Algebraic Approach to Program Verification

北 英彦[†]

坂部 俊樹^{††}

稲垣 康善[†]

Hidehiko KITA

Toshiki SAKABE

Yasuyoshi INAGAKI

[†]名古屋大学 工学部

^{††}三重大学 工学部

[†]Faculty of Engineering, Nagoya University

^{††}Faculty of Engineering, Mie University

あらまし プログラミング言語の形式的仕様記述法の目的の一つは、プログラムの正当性の検証の数学的基礎を与えることである。本論文は、プログラミング言語の代数的仕様記述に基づくプログラムの正当性の検証について述べ、代数的手法に基づくプログラム検証法を明らかにする。まず、プログラムの検証を考慮に入れた場合のプログラミング言語の代数的仕様記述法について述べ、次に、それに基づくプログラムの正当性の定義を与える。そして、一重ループからなるプログラムに対する検証手法と検証例を与える。代数的手法に基づくプログラム検証の利点の一つは、検証プロセスに項書き換えによる計算を組み込むことができ、プログラムの検証を容易にできることである。

Abstract To establish the mathematical foundations for specification and verification of programs is one of the purposes of formal specification of programming languages. In this paper we describe a program verification method based on algebraic specification of programming languages. First taking account of program verification we slightly modify our algebraic specification method of programming languages, and then we define the program correctness based on it. Next, we give a verification method of simple loop programs based on our algebraic method, and show an illustrative example of program verification.

1. まえがき

プログラムの正当性の検証は、プログラミング言語の形式的仕様記述に基づいて行われる必要がある。プログラミング言語の形式的仕様記述法としては、操作的意味論、公理の意味論、表示の意味論、代数的意味論等がある。公理の意味論は、例えば、Hoareの検証体系自身

がプログラミング言語の意味を記述していると見做すもので、検証には適しているが、プログラミング言語の意味論としては、細部の記述ができず、不十分なところがある。そのため、言語の操作的意味論、表示の意味論を与えて、それらに対してHoareの検証体系の無矛盾性を証明する研究がなされている⁽¹⁾。また、言語の表示の意味論そのものに基づく検証方法

も研究されている^{(2), (3)}。

我々は、プログラミング言語の意味論として、代数的意味論について研究を進めている。この方法は、意味領域および意味領域の実現の定義が厳密にできることから、コンパイラの仕様記述、自動生成に適していると考えられ、我々は既に、その仕様記述法を与え⁽⁴⁾、それに基づいて仕様記述例を示し⁽⁵⁾、更に、言語処理系自動生成系の構成に成功している^{(6), (7)}。

プログラミング言語の形式的仕様記述法の目的の一つは、プログラムの正当性の検証の数学的基礎を与えることである。本論文は、プログラミング言語の代数的仕様記述に基づくプログラムの正当性の検証について述べるものであり、代数的手法に基づくプログラム検証法を明らかにする。即ち、まず、プログラムの検証を考慮に入れた場合のプログラミング言語の代数的仕様記述法について説明し、次に、プログラムの正当性の定義を与える。それに基づいて、一重ループからなるプログラムに対する検証手法と検証例を与える。本論文で述べる代数的手法に基づくプログラム検証の特徴の一つは、プログラムの検証過程に項書き換えに基づく計算を組み込むことができ、その結果、意味領域の仕様とプログラムの機能の仕様が項書き換え系としてある条件を満たせば、項の等価性を容易に判定することができ、そのことによって、プログラムの正当性の検証が容易に行なえることである。

2. プログラミング言語の代数的仕様記述法

プログラミング言語の代数的仕様記述法は、言語の構文領域および意味領域を代数（抽象データ型）として定義し、プログラムの意味をこの二つの代数間の準同型写像で与える方法である。

ここで、対象とするプログラミング言語は、入力から出力への関数としてプログラムの意味が捉えられるようなものとする。このようにプログラムの意味を関数と考えて仕様記述を行うわけであるが、抽象データ型の代数的仕様記述法では、汎関数（関数を値とするような関数）を陽に記述する方法がない、即ち、関数自身をデータとして扱えないので、仕様記述を行う際に工夫が必要となる。そこで、関数の代わりに関数の表現に対応するソートを考え、また、関数の表現にその引数を適用する関数を考える。

プログラムの意味について言えば、入力から出力への関数を表すソートと、入力から出力への関数を入力に適用する関数を考え、それらを仕様記述する。

以下、上記の事を考慮に入れて、構文領域、意味領域、意味写像の順にその仕様記述法を定義する。ただし、本報告で用いる抽象データ型に関する概念、記法についての詳細は文献(8)を参照されたい。

2.1. 構文領域の仕様記述法

構文領域の仕様は、基本的には文脈自由文法で与える。ただし、識別子、数字列等は、組み込みの構文領域として扱えるようにする。

【定義 2.1.1】（構文領域の仕様）

構文領域の仕様 G は、5 項組

$$\langle V_T, V^{BL}, V^{NEW}, P, S_0 \rangle$$

である。ここで、

V_T : 終端記号の集合、

V^{BL} : 組み込み非終端記号の集合、

V^{NEW} : 被定義非終端記号の集合、

P : 生成規則の集合、

$S_0 \in V^{NEW}$: 開始記号

である。生成規則は、

$$p : N \rightarrow \alpha$$

の形とする。ここで、

p : 生成規則の名前、

$N \in V^{NEW}$ 、

$$\alpha \in (V^{BL} \cup V^{NEW} \cup V_T)^*$$

である。□

【定義 2.1.2】（組み込み構文領域）

構文領域の仕様を

$$G = \langle V_T, V^{BL}, V^{NEW}, P, S_0 \rangle$$

とする。

組み込み構文領域 $\mathcal{L}^{BL}$ は、集合族

$$\mathcal{L}^{BL} = \langle \mathcal{L}^{BL}_N \rangle \quad N \in V^{BL}$$

である。□

【定義 2.1.3】（構文領域）

構文領域の仕様を

$$G = \langle V_T, V^{BL}, V^{NEW}, P, S_0 \rangle、$$

組み込み構文領域を

$$\mathcal{L}^{BL} = \langle \mathcal{L}^{BL}_N \rangle \quad N \in V^{BL}$$

とする。

このとき、構文領域 $\mathcal{L}$ は、項代数

$$T[G(\mathcal{L}^{BL})]$$

である。□

2.2 意味領域の仕様記述法

意味領域の仕様は、抽象データ型の仕様で与える。ただし、入出力に関するデータ型を陽に指定し、抽象データ型の意味は入出力に関する動作を用いて定める。

【定義 2.2.1】(意味領域の仕様)

意味領域の仕様 $\mathcal{D}$ は、抽象データ型の仕様であり、6 項組

$$\langle \Sigma^{BL}, \Sigma^{IO}, \Sigma, \mathcal{Q}, \mathcal{A}^{IO}, \mathcal{A} \rangle$$

である。ここで、

$$\Sigma^{BL} : \text{シグニチャ } \langle S^{BL}, \mathcal{F}^{BL} \rangle、$$

$$\Sigma^{IO} : \text{シグニチャ } \langle S^{IO}, \mathcal{F}^{IO} \rangle、$$

$$\Sigma : \text{シグニチャ } \langle S, \mathcal{F} \rangle$$

である。ただし、

$$S \supseteq S^{IO} \supseteq S^{BL}、$$

$$\mathcal{F} \supseteq \mathcal{F}^{IO} \supseteq \mathcal{F}^{BL}、$$

$$\text{bool} \in S^{BL}、$$

$$\text{true}, \text{false} \in \mathcal{F}^{BL}, \text{bool}$$

$$\text{input}, \text{output} \in S^{IO}、$$

$$\text{input-output} \in S、$$

$$\text{apply} \in \mathcal{F}^{\text{input-output input, output}}$$

とする。bool はブール型のソート、input はプログラムの入力のソート、output はプログラムの出力のソート、input-output は入力から出力への関数の表現のソート、apply は入力から出力の関数の表現を入力に適用させる関数である。また、

$$\mathcal{Q} : \text{変数集合族 } \langle \mathcal{Q}_s \rangle \ s \in S、$$

$$\mathcal{A}^{IO} : \text{公理の集合}、$$

$$\mathcal{A} : \text{公理の集合}$$

である。ただし、

$$\mathcal{A} \supseteq \mathcal{A}^{IO}、$$

公理は、 $\xi \approx \eta$ の形の等式のみとする。ここで、 ξ, η は、 $\mathcal{A}^{IO}$ の場合は、同一ソートの $\Sigma^{IO}(\mathcal{Q})$ 項、 $\mathcal{A}$ の場合は、同一ソートの $\Sigma(\mathcal{Q})$ 項である。□

【定義 2.2.2】(組み込み意味領域)

意味領域の仕様を

$$\mathcal{D} = \langle \Sigma^{BL}, \Sigma, \mathcal{Q}, \mathcal{A} \rangle$$

とする。

組み込み意味領域 B は、

$$\Sigma^{BL} \text{ 代数}$$

である。ただし、ソート bool に対しては、True, False の 2 元からなる通常のブール型と

する。即ち、 $\text{true}_B = \text{True}$, $\text{false}_B = \text{False}$ とする。□

【定義 2.2.3】(意味領域)

意味領域の仕様を

$$\mathcal{D} = \langle \Sigma^{BL}, \Sigma^{IO}, \Sigma, \mathcal{Q}, \mathcal{A}^{IO}, \mathcal{A} \rangle、$$

B を組み込み意味領域とする。

意味領域の仕様 $\mathcal{D}$ が下記の条件 (1), (2) を満たすとする。

(1) 無矛盾性: 任意のソート $s \in S^{BL}$ 、ソート s の $\Sigma^{BL}(B)$ 項 ξ, η について、

$$\mathcal{A} \models \xi \approx \eta \text{ ならば、} \xi_B = \eta_B$$

である。また、任意のソート $s \in S^{IO}$ 、ソート s の $\Sigma^{IO}(B)$ 項 ξ, η について、

$$\mathcal{A} \models \xi \approx \eta \text{ ならば、} \mathcal{A}^{IO} \models \xi \approx \eta$$

である。

(2) 完全性: 任意のソート $s \in S^{BL}$ 、ソート s の $\Sigma^{IO}(B)$ 項 ξ について、

$$\mathcal{A} \models \xi \approx \eta$$

となる $\Sigma^{BL}(B)$ 項 η が存在する。

このとき、仕様 $\mathcal{D}$ の定める抽象データ型、即ち、意味領域 SD は、項代数 $T[\Sigma(B)]$ の合同関係 $\sim$ による商代数

$$T[\Sigma(B)] / \sim$$

である。ここで、

$$B = \langle B_s \rangle \ s \in S、$$

$$B_s = B_s \quad (s \in S^{BL})$$

$$\emptyset \quad (\text{その他})$$

とする。また、合同関係 $\sim$ は次のように定める。

まず、 $T[\Sigma(B)]$ 上の同値関係の族 $\equiv = \langle \equiv_s \rangle \ s \in S^{IO}$ を、次のように定める。

任意の $\xi, \eta \in T[\Sigma(B)]$ に対して、

$$\xi \equiv_s \eta \text{ 当且 } \mathcal{A} \models \xi \approx \eta$$

ここで、記号 $\models$ 中の B は、組み込み意味領域 B で成り立つ等式を公理として用いてもよいことを意味する。

そして、この $\equiv$ から、 $\sim = \langle \sim_s \rangle \ s \in S$ を次のように定める。即ち、すべての $s \in S^{IO}$ について、 $\sim_s = \equiv_s$ を満たす項代数 $T[\Sigma(B)]$ 上の最大の合同関係とする。□

2.3 意味写像の仕様記述法

既に、報告した我々の代数的仕様記述法⁽⁴⁾では、意味写像は文脈自由文法の各生成規則に意味方程式を一つ与えることで仕様記述される。本論文でも、基本的には同じであるが、組み込み構文領域に対する意味は、組み込み意味写像で、既に、与えられているものとして意味写像

の仕様の定義を行う。

【定義 2.3.1】(意味写像の仕様)

構文領域の仕様を

$$G = \langle V_T, V^{BL}, V^{NEW}, P, S_0 \rangle,$$

意味領域の仕様を

$$D = \langle \Sigma^{BL}, \Sigma^{IO}, \Sigma, Q, A^{IO}, A \rangle,$$

組み込み構文領域を

$$\mathcal{L}^{BL} = \langle \mathcal{L}^{BL}_N \rangle N \in V^{BL},$$

組み込み意味領域を

$$\Sigma^{BL} \text{代数 } B$$

とする。

意味領域の仕様 Γ は、5 項組

$$\langle D, M^{BL}, M^{NEW}, X, R \rangle$$

である。ここで、

$D: V^{BL} \cup V^{NEW} \rightarrow S$: 非終端記号に意味領域のソート割り当てる関数、ただし、 $N \in V^{BL}$ について、 $D(N) \in S^{BL}$

である。ただし、

開始記号 $S_0 \in V^{NEW}$ に対して、

$$D(S_0) = \text{input-output}$$

である。また、

$M^{BL} = \{M_N \mid N \in V^{BL}\}$: 各組み込み非終端記号 N に対する組み込み構文領域 $\mathcal{L}^{BL}_N$ から組み込み意味領域 $B_{D(N)}$ への関数集合上の変数 (関数変数) M_N の集合

$M^{NEW} = \{M_N \mid N \in V^{NEW}\}$: 構文領域 $T[G(\mathcal{L}^{BL})]_N$ から意味領域 $S_{D(N)}$ への関数集合上の関数変数 M_N の集合

$X = \langle X_N \rangle N \in V$: 構文領域上の変数集合族

$R = \{R_p \mid p \in P\}$: 意味方程式の集合である。意味方程式は、各生成規則

$$p: N \rightarrow \alpha, nt(\alpha) = N_1 \cdots N_n$$

に対して、

$$M_N[p(x_1, \dots, x_n)] = \xi_{SD}(M_{N_1}[x_1], \dots, M_{N_n}[x_n])$$

の形とする。ここで、

nt : 記号列中の非終端記号を取出す関数、

$$x_i \in X_{N_i} \quad (i=1, \dots, n),$$

ξ_{SD} : ソート $D(N)$ の $\Sigma(B)$ 項 ξ の意味領域 S_D 上での導出関数

である。□

【定義 2.3.2】(組み込み意味写像)

$\Gamma = \langle D, M^{BL}, M^{NEW}, X, R \rangle$ を意味写像の仕様、 $\mathcal{L}^{BL} = \langle \mathcal{L}^{BL}_N \rangle N \in V^{BL}$ を組み込

み構文領域、 Σ^{BL} 代数 B を組み込み意味領域とする。

組み込み意味写像 M^{BL} は、写像族

$$M^{BL} = \langle M^{BL}_N: \mathcal{L}^{BL}_N \rightarrow B_{D(N)} \rangle N \in V^{BL}$$

である。□

【定義 2.3.3】(意味写像)

$\Gamma = \langle D, M^{BL}, M^{NEW}, X, R \rangle$ を意味写像の仕様、 $M^{BL}_N = \langle M^{BL}_N: \mathcal{L}^{BL}_N \rightarrow B_{D(N)} \rangle N \in V^{BL}$ を組み込み意味写像とする。

意味写像 M は、次の条件 (1)(2) を満たす写像族

$$M = \langle M_N: T[G(\mathcal{L}^{BL})]_N \rightarrow S_{D(N)} \rangle N \in V^{NEW} \cup V^{BL}$$

である。

(1) $N \in V^{BL}$ について、 $M_N = M^{BL}_N$

(2) 各意味方程式 R_p について、

$$M_N[p(x_1, \dots, x_n)] = \xi_{SD}(M_{N_1}[x_1], \dots, M_{N_n}[x_n]) \quad \square$$

意味写像の定義より、プログラムの意味は入力から出力への関数の表現となっている。即ち、

$$M_{S_0}: \mathcal{L}_{S_0} \rightarrow S_{D(S_0)}$$

である。

3. プログラムの正当性

2. で定義した代数的仕様記述法によって、プログラミング言語の意味を形式的に与えることができる。即ち、プログラムの意味は、入力から出力への関数の表現として仕様記述できる。

プログラムの正当性には、プログラミング言語の意味論によって、細部が異なる種々の定義があるが、ここでは、直観的にいうと、入力条件を満たすすべての入力に対して、プログラムの機能の仕様が定義する出力関数の値と、その入力にプログラムの意味である関数を適用した結果が等しいことであると定義する。まず、プログラムの機能の仕様を抽象データ型の仕様を用いて形式的に定義する。

【定義 3.1】(プログラムの機能の仕様)

プログラムの機能の仕様は、意味領域の仕様と同様な抽象データ型の仕様

$$\langle \Sigma^{BL}, \Sigma^{IO}, \Sigma', Q', A^{IO}, A' \rangle$$

である。ただし、 $\Sigma^{BL}, \Sigma^{IO}, A^{IO}$ は意味領域の仕様のもとの共通とする。また、 Σ の中に次の二つの関数記号があるとする。

$$\text{入力条件 } Q: \text{input} \rightarrow \text{bool}$$

出力関数 $S : \text{input} \rightarrow \text{output}$

ここで、 $\text{input}, \text{output} \in S^{10}$ であり、 input はプログラムの入力ソート、 output はプログラムの出力ソートである。□

【定義 3.2】(プログラムの機能)

プログラムの機能の仕様を

$$(\Sigma^{BL}, \Sigma^{10}, \Sigma', Q', A^{10}, A')$$

とする。

仕様の定義するプログラムの機能 F は、仕様
が次の条件(1)、(2)を満たすとき、抽象データ型

$$T[\Sigma'(\bar{B})] / \sim'$$

である。この抽象データ型の定義は、意味領域
の場合と同じである。

(1) 無矛盾性：任意のソート $s \in S^{BL}$ 、ソート
 s の $\Sigma^{BL}(\bar{B})$ 項 ξ 、 η について、

$$A \models \xi \approx \eta \text{ ならば、} \xi_B = \eta_B$$

である。また、任意のソート $s \in S^{10}$ 、ソート
 s の $\Sigma^{10}(\bar{B})$ 項 ξ 、 η について、

$$A \models \xi \approx \eta \text{ ならば、} A^{10} \models \xi \approx \eta$$

である。

(2) 完全性：任意のソート $s \in S^{BL}$ 、ソート s
の $\Sigma^{10}(\bar{B})$ 項 η について、

$$A \models \xi \approx \eta$$

となる $\Sigma^{BL}(\bar{B})$ 項 η が存在する。

また、任意のソート $s \in S^{10}$ 、ソート s の $\Sigma(\bar{B})$
項 ξ について、

$$A \models \xi \approx \eta$$

となる $\Sigma^{10}(\bar{B})$ 項 η が存在する。□

プログラムの機能の中で特に意味のあるのは、

$$\text{入力条件 } Q_F : F_{\text{input}} \rightarrow F_{\text{bool}}$$

$$\text{出力関数 } S_F : F_{\text{input}} \rightarrow F_{\text{output}}$$

である。この二つの関数で、実質的にプログラ
ムの機能を定義している。

このようにプログラムの機能の仕様を、抽象
データ型の仕様として定義し、このプログラ
ムの機能に対するプログラムの正当性の定義を次
のように行う。

【定義 3.2】(プログラムの正当性)

プログラムの機能の仕様を

$$(\Sigma^{BL}, \Sigma^{10}, \Sigma', Q', A^{10}, A')$$

として、 F を仕様の定義するプログラムの機能
とする。また、意味領域の中で、入力から出力
への関数を入力に適用する関数を apply_{SD} とす
る。

プログラム P がプログラムの機能に対して、

正当であるとは次の条件が成立することである。

$$(a) \forall x \in F_{\text{input}}, Q_F(x) \rightarrow$$

$$\text{apply}_{SD}(M_{So}[P], h(x)) =_{10} S_F(x)$$

関数 $h : F_{\text{input}} \rightarrow S D_{\text{input}}$ は、

$$h(\sim'[\xi]) = \sim[\eta]$$

$$(\xi \sim \eta, \eta \in T[\Sigma^{10}(\bar{B})])$$

によって、また、2項関係

$$=_{10} \subseteq S D_{\text{output}} \times F_{\text{output}}$$

は、

$$\sim[\xi] =_{10} \sim'[\eta]$$

$$\text{iff } \sim[\xi]^{10} = \sim'[\eta]^{10}$$

によって定める。ここで、 $\xi \in [\Sigma(\bar{B})]$ 、
 $\eta \in T[\Sigma'(\bar{B})]$ に対して、

$$\sim[\xi] = \{\zeta \in T[\Sigma(\bar{B})] \mid \xi \sim \zeta\}$$

$$\sim'[\eta] = \{\zeta \in T[\Sigma'(\bar{B})] \mid \eta \sim' \zeta\}$$

$$\sim[\xi]^{10} = \{\zeta \in T[\Sigma^{10}(\bar{B})] \mid \xi \sim \zeta\}$$

$$\sim'[\eta]^{10} = \{\zeta \in T[\Sigma^{10}(\bar{B})] \mid \eta \sim \zeta\}$$

である。□

4. 一重ループプログラムの検証

上で定義したプログラムの正当性に基づいて
一重ループプログラムの検証方法について述べ
る。

ここで、プログラムの正当性の検証が行い易
いように意味領域の仕様とプログラムの機能の
仕様に条件をつけ、プログラムの正当性の定義
をより検証し易い形に言い換える。

条件(1)： 意味領域の仕様とプログラムの機能
の仕様は項書き換え系⁽⁹⁾として、合流性を持
つものとする。任意の $s \in S^{10}$ について、ソ
ート s の $\Sigma(\bar{B})$ 項、 $\Sigma'(\bar{B})$ 項は、正規形を持つ
ならば、その正規形は $\Sigma^{10}(\bar{B})$ 項であるとする。
また、任意の $s \in S^{10}$ について、ソート s の $\Sigma^{10}(\bar{B})$
項は、 $\Sigma^{10}(\bar{B})$ 項を正規形として持つとす
る。□

このように条件(1)を仮定すると、 F_{input} の
要素である同値類の代表元として、正規形を用
いることができる。そして、入力条件を満たす
入力に対して、プログラムの意味とプログラムの
機能が等しいことも各々の正規形が等しいこ
とと等価になる。即ち、プログラムの正当性を
次のように言い換えることができる。

(b) $\forall x \in NF_{input}$,
 $nf(Q(x)) = nf(true()) \rightarrow$
 $nf(apply(Mso[P], x))$
 $= nf(S(x))$

ここで、

NF_{input} : ソートinput の正規形の集合、
 $nf(\xi)$: 項 ξ の正規形、
 $Mso[P]$: プログラム P を意味写像の仕
 様に従って変換した $\Sigma(\bar{B})$ 項

を表す。

次に、プログラムの正当性の検証の例として、
 プログラミング言語 PL/0.2 + ⁽⁵⁾ で書かれ
 た一重ループプログラムに上で述べた検証手法
 を適用する。

プログラミング言語 PL/0.2 + のプログラ
 ムの意味は、ファイルからファイルへの関数の
 表現として記述されている。PL/0.2 + の仕
 様を以下に記述するのは、不可能なので、仕様
 については、文献(5)を参照されたい。

この言語 PL/0.2 + で書かれた検証に用い
 る一重ループプログラムを図1に与える。

```
var in, count;
begin
  read(count);
  read(in);
  while not eof do
    begin
      read(in);
      count := count + 1;
      write(count, in)
    end
  end.
```

図1 PL/0.2 + の一重ループプログラム

次にプログラムの機能の仕様を図2に与える。
 このプログラムの機能は、入力ファイルから整
 数 x を読んできて、その数がファイル中の何番
 目の数かという値 n とその整数 x を出力ファ
 イルに書き出す関数である。ただし、最初に入
 力ファイルから整数を、変数の初期化をする。

$\Sigma^{BL} = (S^{BL}, \mathcal{F}^{BL})$
 $S^{BL} = \{ \text{bool}, \text{int} \}$ $\mathcal{F}^{BL} = \dots$
 $\Sigma^{I0} = (S^{I0}, \mathcal{F}^{I0})$
 $S^{I0} = \{ \text{file} \} \cup S^{BL}$ $\mathcal{F}^{I0} = \mathcal{F}^{BL} \cup \dots$
 $\Sigma = (S, \mathcal{F})$
 $S = S^{I0}$
 $\mathcal{F} = \mathcal{F}^{I0} \cup$

{ spec : file $\rightarrow$ file,
 input : file $\rightarrow$ bool,
 count : file file int $\rightarrow$ file }

$Q = \{ \text{int0}, \text{int1} : \text{int}, \text{file0}, \text{file1} : \text{file} \}$

$\mathcal{A}^{I0} = \dots$

$\mathcal{A} = \mathcal{A}^{I0} \cup$

{ input(empty_file()) $\approx$ false(),
 { input(write_file(empty_file(), int0))
 $\approx$ false(),
 input(write_file(
 write_file(file0, int0), int1))
 $\approx$ true(),
 spec(file0, file1)
 $\approx$ count(remove_data(file0),
 file1,
 read_file(file0)),
 count(empty_file(), file0, int0)
 $\approx$ file0,
 count(write_file(file0, int0),
 file1, int1)
 $\approx$ count(remove_data(
 write_file(file0, int0)),
 write_file(
 write_file(
 file1,
 sum_int(int1, one()))),
 sum_int(int1, one())) }

図2 プログラムの機能の仕様

さて、この PL/0.2 + の意味領域の仕様と
 プログラムの機能の仕様は、上述した条件(1)
 を満たしている。そこで、プログラムの正当性
 を検証するのに、(b)の方の定義を用いること
 ができる。(b)を PL/0.2 + の仕様記述と一重
 ループプログラムのプログラムの機能の仕様
 に合わせて書き直す次のようになる。

(c) $\forall x \in NF_{file}$,
 $nf(input(x)) = nf(true()) \rightarrow$
 $nf(run(Mso[P], x))$
 $= nf(spec(x))$

この(c)を証明する代わりに次の(d)を証明し、その系として(c)が成立することを導く。

```
(d)  $\forall x, y \in NF_{file},$   

 $nf(input(x)) = nf(true()) \rightarrow$   

 $nf(run2(Mso[P], x, y))$   

 $= nf(spec2(x, y))$ 
```

ここで、run2、spec2 の第 2 引数は、プログラムを実行する前の出力ファイルを表すものとする。即ち、

```
run(Mso[P], x))  

 $\approx run2(Mso[P], x, empty\_file())$   

spec(x))  

 $\approx spec2(x, empty\_file())$ 
```

とする。

この(d)を入力の変数に関する帰納法を用いて証明する。入力条件を満たす入力、人手で与えることにする。即ち、帰納法の基底の入力、帰納法の仮定の入力、帰納法の帰納段階の入力を人手で与える(図3)。これらの入力による帰納法が入力条件を満たす、すべての入力を尽くしていることは、簡単に示すことができる。

(* 帰納法の基底の入力 *)

```
write_file(  

write_file(empty_file(), i), c)
```

(* 帰納法の仮定の入力 *)

```
write_file(write_file(file0, i), c)
```

(* 帰納法の帰納段階の入力 *)

```
write_file(  

write_file(  

write_file(file0, x), i), c)
```

file0 : ソートfileの変数,

c, i, x : ソートintの変数

図3 帰納法で用いる入力

これらの入力を用いた帰納法を各段階を、以下に示す。

(* 帰納法の基底段階 *)

```
nf(spec2(write_file(  

write_file(empty_file(), i),  

c), file1)) = file1
```

```
nf(run2(Mso[P],  

write_file(  

write_file(empty_file(), i),  

c), file1)) = file1
```

故に、基底段階については、成立する。

(* 帰納法の仮定段階 *)

```
nf(spec2(write_file(  

write_file(file0, i), c)  

file1))  

= count(file0, file1, c) ... ( $\alpha$ )  

nf(run2(Mso[P],  

write_file(  

write_file(file0, i), c)  

file1))  

= make_file(  

retrieve(  

apply_switch(not.eof(file0)),  

composition(iterate(xxx), yyy),  

i_state-state(),  

add_id(add_id(add_id(add_id(  

init_state(),  

input, make_attr_file(file0)),  

output, make_attr_file(file1)),  

in, make_attr_var(i)),  

count, make_attr_var(c))),  

output) ... ( $\beta$ )  

( $\alpha$ )  $\approx$  ( $\beta$ ) が成立するとする。
```

(* 帰納法の帰納段階 *)

```
nf(spec2(write_file(  

write_file(  

write_file(file0, x), i), c)  

file1))  

= count(file0,  

write_file(  

write_file(file1,  

sum_int(c, 1)),  

x), sum_int(c, 1)) ... ( $\gamma$ )  

nf(run2(Mso[P],  

write_file(  

write_file(  

write_file(file0, x), i), c)  

file1))  

= make_file(  

retrieve(  

apply_switch(not.eof(file0)),  

composition(iterate(xxx), yyy),
```

```

i_state=state(),
add_id(add_id(add_id(add_id(
    init_state(),
    input, make_attr_file(file0)),
    output, make_attr_file(
        write_file(
            write_file(file1,
                sum_int(c, 1)),x)
    in, make_attr_var(x)),
    count, make_attr_var(c))),
output) ... ( $\delta$ )

```

式 (α), (β) 中の変数 file1, c, i にそれぞれ、

```

write_file(
    write_file(file1,sum_int(c, 1)),x)
sum_int(c, 1)
x

```

を代入すると式 (r), (δ) になる。

(α) $\approx$ (β) より、(r) $\approx$ (δ)

以上で、帰納的段階を証明できた。故に、(d) が成立する。(d)より(c)が成立することは、明らかである。即ち、図1の一重ループプログラムは、そのプログラムの機能に対して、正当であると検証された。

このように、意味領域の仕様、プログラムの機能の仕様が項書き換え系としてある条件(1)を満たすとプログラムの正当性の定義を(b)のように言い換えることができ、プログラムの正当性の検証に項書き換え系を活用することができる。

5. まとめ

プログラムの正当性の検証を考慮に入れて、プログラミング言語の代数的仕様記述法を定義し、それに基づき、プログラムの正当性の定義を行なった。また、一重ループプログラムの検証方法とその実行例を示した。

一般のプログラムの検証については、各ループごとに、その仕様を与え、部分プログラムの検証を行い、それを用いて全体のプログラムの検証を行えばよい。

問題点および今後の課題としては、プログラミング言語の代数的仕様記述法では、意味領域は、抽象データ型であるため、汎関数を陽に用いることができない。そのため、プログラムの意味を入力から出力への関数と捉えて仕様記述するには工夫が必要で、その結果、プログラム

の正当性の定義がやや技巧的なものになっている。プログラムの正当性の定義を素直なものにするためには、プログラムの意味を関数そのものとして扱う必要がある。そのためには、抽象データ型において汎関数が陽に扱えるようにする必要がある。

ここで述べたプログラムの正当性の定義はすべてのの入力に対して停止するプログラムのみを対象にする。止まらない可能性のあるプログラムについては、部分代数の概念を用いて、意味領域の定義を行い、プログラムの意味を部分関数として取り扱う必要があり、今後の課題である。

謝 辞

日頃御指導下さる豊橋技術科学大学本多波雄学長、名古屋大学福村晃夫教授、並びに御討論下さる研究室の皆様へ感謝致します。なお、この研究は一部、文部省科研費（一般(c)課題番号60550263 および特定研究(1)多元知識情報 課題番号61101003)の援助を受けた。

文 献

- (1) Donahue, J.E. : "Complementary Definitions of Programming Language Semantics", Lecture Note in Computer Science 42 (1976)
- (2) Aiello, L., Aiello, M., Weyhrauch, R.W. : "Pascal in LCF : Semantics and Examples of Proof", Theor. Comput. Sci., Vol.5, pp.135-177 (1977)
- (3) Polak, W. : "Program Verification Based on Denotational Semantics", 8th Annual ACM Symp. on POPL., pp.149-158 (1981)
- (4) 北他 : 「抽象データタイプに基づく形式言語の意味記述」, 信学技報, AL83-23 (1983)
- (5) 高木他 : 「入出力機能とパラメタ付手続きを付加したプログラミング言語 PL/O の代数的仕様記述」, 信学技報, AL85-44 (1985)
- (6) 川辺他 : 「抽象データ型の直接実現システム」, 信学技報, AL83-65 (1984)
- (7) 酒井他 : 「プログラミング言語の代数的仕様記述からのコンパイラ自動生成」, 信学技報, AL85-10 (1985)
- (8) 稲垣, 坂部 : 「抽象データタイプの代数的仕様記述法の基礎(1), (2), (3), (4)」, 情報処理, Vol.25, No.1, No.5, No.7 No.9 (1984)
- (9) 二木他 : 「項書き換え型計算モデルとその応用」, 情報処理, Vol.24, No.2 (1983)