数理解析研究所講究録
第381巻 1980年 258-275

不完全指定形抽象データタイプの仕様記述と実現

坂部俊樹† 稲垣康善†† 本多波雄†
(†名大・工　　††三重大・工)

1 まえがき　抽象データタイプは，信頼性が高く効率的でありかつ柔軟性のあるソフトウェアの開発には欠かせない重要な概念として注目され，特に，その形式的仕様記述を与える代数的手法について多くの研究がなされている[1~7]．代数的仕様記述の基本的な考え方は，抽象データタイプを多sort代数として捉え，公理(等式)の有限集合で与えられる仕様によって記述される抽象データタイプは公理を満たす代数のカテゴリの始代数であるとするものである．従来，このような議論の中で取り扱われる代数の演算はすべて全域的に定義される関数であった．そのために，抽象データタイプの演算のうち未定義部分があってもよい演算も，UNDEFINEDまたはERRORという特別な値を用いて，全域的な関数として取り扱われ，そこに多少の不自然さが見うけられた．そこで本報告では，まずはじめに演算が一般に部分関数であるような

抽象データタイプ（これを不完全指定形抽象データタイプと呼ぶ）を導入して，その代数的仕様記述の方法を明らかにする．この方法によれば，実際には起こり得ない演算の組合せ等を未定義部分のある演算で表現することによって多くの場合に従来の仕様記述と比べて簡潔な仕様記述が得られる．さらに，不完全指定形抽象データタイプの演算の未定義部分を効果的に利用する実現を定式化し，従来のモデルでしばしば見られたむだな条件判断（IF THEN ELSE 演算の重複）を自然に除去できるなど，より効率の良い実現が得られることを示す．

2　部分関数を演算とする多sort代数　この章では，本報告で中心的役割を果たす演算が部分関数である多sort代数およびそのカテゴリについて簡単に説明する．本報告では，Goguen 他 [1] の流儀に沿って議論が進められるので，なるべく [1] と共通の記法を用いるものとし，それらについての詳しい定義等は省略することがある．

sortの集合を S，S-sorted operator domain を $\Sigma = \langle \Sigma_{w,s} \rangle_{w \in S^*, s \in S}$ とする．演算の合成を表すsort s の Σ 項の集合を $T_{\Sigma,s}$ と書き，変数の入っているsort s の Σ 項（$\Sigma(X)$ 項と呼ぶ）の集合を $T_\Sigma(X)_s$ と書く．ただし X は変数集合族 $\langle X_s \rangle_{s \in S}$ である．$t \in T_\Sigma(X)_s$ の中の変数が $x_1, \cdots, x_n$ のとき，これを明確にするために $t(x_1, \cdots, x_n)$ と書くことがある．また，t を木で表すこと

ができるので，tの深さ$|t|$をその木の高さで定義する．Σ代数Aは集合族$\langle A_s\rangle_{s\in S}$と各演算記号$\sigma\in\Sigma_{w,s}$（$w=s_1\cdots s_n$）に対する部分関数$\sigma_A: A_{s_1}\times A_{s_2}\times\cdots\times A_{s_n}\to A_s$とからなる．ただし，$\sigma\in\Sigma_{\lambda,s}$（$\lambda$は空系列）に対しては，$\sigma_A\in A_s$とする．注意すべき点は，[1,3,7]等で使われる代数と異なり，演算が部分関数となっていることである．これに伴ない，合成演算を表すΣ(X)項tのA上での解釈t_A（tによって定められるA上の合成演算（derived operation）という．）も部分関数となる．すなわち，t_Aの定め方は関数の引数のどれか1つでも未定義ならその関数は未定義とする通常の部分関数合成の定義に基づく．

Σ代数のカテゴリはΣ代数間の準同形写像を定めることによって与えられる．すべての演算が全域的であるΣ代数に対する準同形写像の定義をそのまま流用して，演算が部分関数であるΣ代数の準同形写像を定義し，それをΣ準同形写像と呼ぶ．すなわち，Σ代数A,Bに対して，次の条件(h0),(h1)を満たす写像族$h=\langle h_s: A_s\to B_s\rangle_{s\in S}$をΣ準同形写像といい，$h: A\to B$と書く．

(h0)　$\sigma\in\Sigma_{\lambda,s}$ならば　$h_s(\sigma_A)=\sigma_B$．

(h1)　$\sigma\in\Sigma_{s_1\cdots s_n,s}$ならば　$h_s\circ\sigma_A=\sigma_B\circ(h_{s_1}\times\cdots\times h_{s_n})$．ただし，$\circ$は部分関数の合成であり，写像$h_{s_1}\times\cdots\times h_{s_n}: A_{s_1}\times\cdots\times A_{s_n}\to B_{s_1}\times\cdots\times B_{s_n}$は$h_{s_1}\times\cdots\times h_{s_n}(a_1,\cdots,a_n)=(h_{s_1}(a_1),\cdots,h_{s_n}(a_n))$で定めら

れる．Σ準同形写像 $h: A \to B$ と $h': B \to C$ の合成 $h' \circ h: A \to C$ を $\langle h'_s \circ h_s \rangle_{s \in S}$ で定義する．

このとき次の命題が得られる．すべてのΣ代数のクラスを Alg_Σ，すべてのΣ準同形写像のクラスを $\mathcal{H}_\Sigma$（あるいは単に $\mathcal{H}$）とするとき，

命題2.1　$(Alg_\Sigma, \mathcal{H}_\Sigma)$はカテゴリである．以下では，このカテゴリを$\mathcal{H}_\Sigma$で表すことにする．

Σ準同形写像は不完全指定形抽象データタイプを導入して自由度のある実現を考察するという我々の目標にとってはやや制限が強いので，Σ準同形写像より制限の弱い準同形写像を次に定義する．

任意の非負整数をnとする．以下では非負整数の集合をωと書く．Σ代数A, Bに対して，条件(h0)および次の条件(h2), (h3)を満たす写像の族 $h = \langle h_s : A_s \to B_s \rangle_{s \in S}$ をn準同形写像という．

(h2)　$\sigma \in \Sigma_{s_1 \cdots s_m, s}$ ならば　$h_s \circ \sigma_A \subseteq \sigma_B \circ (h_{s_1} \times \cdots \times h_{s_m})$．

(h3)　$t(x_1, \cdots, x_m) \in T_\Sigma(X)_s$, $x_i \in X_{s_i}$ かつ $|t| \leq n$ ならば，$h_s \circ t_A \supseteq (h_s(A_s) \restriction t_B \restriction h_{s_1}(A_{s_1}) \times \cdots \times h_{s_m}(A_{s_m})) \circ (h_{s_1} \times \cdots \times h_{s_m})$．ただし，$h_s(A_s) \restriction t_B \restriction h_{s_1}(A_{s_1}) \times \cdots \times h_{s_m}(A_{s_m})$ は部分関数 $t_B : B_{s_1} \times \cdots \times B_{s_m} \to B_s$ の定義域を $h_{s_i}(A_{s_i}) = \{h_{s_i}(a) \mid a \in A_{s_i}\}$ $(\subseteq B_{s_i})$ の直積へ制限し，値域を $h_s(A_s)$ $(\subseteq B_s)$ へ制限した部分関数である．

すべての n 準同形写像のクラスを $n\text{-}\mathcal{H}_\Sigma$（あるいは単に $n\text{-}\mathcal{H}$）で表す．特に，0準同形写像，すなわち (h0) と (h2) を満たす $\langle h_\delta \rangle_{\delta\in S}$ を弱準同形写像と呼ぶことがある．また，任意の $n\in\omega$ に対して (h3) が成り立つ弱準同形写像を ∞ 準同形写像という．さらに，すべての $\delta\in S$ に対して $h_\delta : A_\delta \to B_\delta$ が全(単)射である n 準同形写像 h を n 全(単)準同形写像といい，そのクラスを $n\text{-}\mathcal{H}_\Sigma^{onto}$（$n\text{-}\mathcal{H}_\Sigma^{1:1}$）で表す．

このように定義される n 準同形写像について次の性質がある．

性質2.1　適当に Σ を選ぶと，任意の $n\in\omega$ に対して

$$n\text{-}\mathcal{H}_\Sigma^{1:1} \supsetneq (n+1)\text{-}\mathcal{H}_\Sigma^{1:1} \supsetneq \infty\text{-}\mathcal{H}_\Sigma^{1:1}$$　（$\supsetneq$ は真の包含関係）

性質2.2　任意の Σ に対して

$$1\text{-}\mathcal{H}_\Sigma^{onto} = 2\text{-}\mathcal{H}_\Sigma^{onto} = \cdots = \infty\text{-}\mathcal{H}_\Sigma^{onto}$$

性質2.3　適当に Σ を選ぶと，$0\text{-}\mathcal{H}_\Sigma \supsetneq \mathcal{H}_\Sigma \supsetneq 1\text{-}\mathcal{H}_\Sigma^{onto}$

性質2.4　$0\text{-}\mathcal{H}_\Sigma$，$n\text{-}\mathcal{H}_\Sigma^{1:1}$ $(n\in\omega)$，$\infty\text{-}\mathcal{H}_\Sigma^{1:1}$，$0\text{-}\mathcal{H}_\Sigma^{onto}$，$1\text{-}\mathcal{H}_\Sigma^{onto}$ は合成のもとで閉じている．

性質2.5　$n\text{-}\mathcal{H}_\Sigma$ $(n\geq 1)$ および $\infty\text{-}\mathcal{H}_\Sigma$ は合成のもとで閉じていない．

性質2.4，2.5 から本報告で新たに定めた n 準同形写像の一部のクラスだけが Σ 代数のカテゴリを形成することが知られる．すなわち，

命題2.2　$0\text{-}\mathcal{H}_\Sigma$，$n\text{-}\mathcal{H}_\Sigma^{1:1}$ $(n\in\omega)$，$\infty\text{-}\mathcal{H}_\Sigma^{1:1}$，$0\text{-}\mathcal{H}_\Sigma^{onto}$，$1\text{-}\mathcal{H}_\Sigma^{onto}$ は

カテゴリである.

3 不完全指定形抽象データタイプの仕様記述　本章では，まず，不完全指定形抽象データタイプの仕様記述の形式的定義を与え，そのあとで，抽象データタイプの典型的な例であるStackの仕様記述を例としてとりあげる．尚以下では，特に断わらない限り，仕様記述と言えば不完全指定形の仕様記述を意味する．

Σ を S-sorted-operator domain とする．

定義3.1　Σ等式は対 $e=(u,v)$ である．ただし，$\Sigma(X)$項 u, vは同一のsortである．以下では e を $u=v$ と書く．Σ代数 A は $u_A=v_A$ である時かつその時に限り e を満足するという．Σ等式の集合 E に対して，A がすべての E の元を満足する時かつその時に限り A は E を満足するという．

従来の完全指定形の仕様記述は，このΣ等式の有限集合で定義されていたが，このままでは演算の定義域に関する情報が不足で，不完全指定形抽象データタイプの仕様記述としては不十分である．そこで次の定義を与える．

定義3.2　Σ代数を A とする．Σ項 $t\in T_{\Sigma,s}$ は，$t_A\in A_s$ の時かつその時に限り A で定義されるという．Σ項の集合 $\Delta=\bigcup_{s\in S}\Delta_s$ ($\Delta_s\subseteq T_{\Sigma,s}$) は Δ のすべての元が A で定義されるときかつその時に限り A で定義されるという．

定義 3.3　不完全指定形抽象データタイプの仕様記述は3項組 (Σ, E, Δ) である．ここに，Σ は S-sorted operator domain，E は Σ 等式の有限集合，Δ は Σ 項の集合である．

さて，このように定義された仕様記述の意味，すなわち，仕様記述によって記述される不完全指定形抽象データタイプを次のように定める．

定義 3.4　仕様記述 (Σ, E, Δ) によって記述される不完全指定形抽象データタイプは，E を満足しかつ Δ が定義されるすべての Σ 代数のクラスによって定められる $0\text{-}\mathcal{H}_\Sigma$ の部分カテゴリ $0\text{-}\mathcal{H}_{\Sigma,E,\Delta}$ の始代数である．

仕様記述の意味を直観的に理解するために，$0\text{-}\mathcal{H}_{\Sigma,E,\Delta}$ の始代数の具体的な構成を与える．そのために次の定義を与える．

定義 3.5　Σ 代数 A 上の Σ 合同 $\equiv$ は次の条件を満たす各 $s \in S$ に対する A_s 上の同値関係 $\equiv_s$ からなる族 $\langle \equiv_s \rangle_{s \in S}$ である．$\sigma \in \Sigma_{s_1 \cdots s_n, s}$，$a_i \equiv_{s_i} a_i'$ $(i = 1, \cdots, n)$ かつ σ_A が $(a_1, \cdots, a_n)$ に定義されるならば σ_A は $(a_1', \cdots, a_n')$ にも定義され，かつ，$\sigma_A(a_1, \cdots, a_n) \equiv_s \sigma_A(a_1', \cdots, a_n')$．$a \in A_s$ の $\equiv_s$ による同値類を $[a]$ と書く．Σ 合同 $\equiv$ に関する A の商代数 $A/\equiv$ は通常のように定義される．すなわち，(1) 各 $s \in S$ に対して $(A/\equiv)_s = A_s/\equiv_s$．(2) 各 $\sigma \in \Sigma$ に対して，(i) $\sigma \in \Sigma_{\lambda, s}$ ならば $\sigma_{A/\equiv} = [\sigma_A]$，(ii) $\sigma \in \Sigma_{s_1 \cdots s_n, s}$ ならば，$[a_i] \in (A/\equiv)_{s_i}$ $(i = 1, \cdots, n)$ に対して，

$$\sigma_{A/\equiv}([a_1],\cdots,[a_n]) = \begin{cases} [\sigma_A(a_1,\cdots,a_n)\ ;\ \sigma_A \text{ が } (a_1,\cdots,a_n) \text{ に定義されるとき} \\ \text{未定義}\ ;\ \text{その他} \end{cases}$$

定義 3.6　$\langle T_{\Sigma,s}\rangle_{s\in S}$ 上で各演算を次のように定めて得られる Σ 代数を T_Σ とする．各々の $\sigma\in\Sigma$ に対して，

(i) $\sigma\in\Sigma_{\lambda,s}$ ならば $\sigma_{T_\Sigma} = \sigma\ (\in T_{\Sigma,s})$

(ii) $\sigma\in\Sigma_{s_1\cdots s_n,s}$ かつ $t_i\in T_{\Sigma,s_i}$ ならば $\sigma_{T_\Sigma}(t_1,\cdots,t_n) = \sigma(t_1,\cdots,t_n)$

定義 3.7　Σ 等式の集合 E によって生成される Σ 代数 T_Σ 上の Σ 合同を $\overset{E}{\equiv}$ とする．ここで，(Σ,E) を従来の完全指定形の仕様記述とみなすと，それによって記述される完全指定形抽象データタイプは $T_\Sigma/\overset{E}{\equiv}$（以下では $T[\Sigma,E]$ と書く）であることに注意しよう．

さて，仕様記述 (Σ,E,Δ) に対して次のような Σ 代数 $U[\Sigma,E,\Delta]$（あるいは単に U）を構成する．各 $s\in S$ に対して，集合 $U_s \subseteq T[\Sigma,E]_s\ (= T_{\Sigma,s}/\overset{E}{\equiv_s})$ を

$$U_s = \{[a]\in T_{\Sigma,s}/\overset{E}{\equiv_s} \mid \exists s'\in S,\ \exists t(x)\in T_\Sigma(X)_{s'},\ \exists b\in\Delta_{s'}\cup\Sigma_{\lambda,s'},\ x\in X_s \text{ かつ } t(a)\overset{E}{\equiv_{s'}} b\}$$

で定める．各演算記号 σ に対して，

(1) $\sigma\in\Sigma_{\lambda,s}$ ならば $\sigma_U = [\sigma]$，

(2) $\sigma\in\Sigma_{s_1\cdots s_n,s}$ かつ $[a_i]\in U_{s_i}$ $(i=1,\cdots,n)$ ならば，

$$\sigma_U([a_1],\cdots,[a_n]) = \begin{cases} [\sigma(a_1,\cdots,a_n)]\ ;\ [\sigma(a_1,\cdots,a_n)]\in U_s \\ \text{未定義}\ ;\ \text{その他} \end{cases}$$

と定める．この定義が正当でUがΣ代数であることは容易に確められる．構成から明らかなように，Uのsort sの台U_sは，定数もしくはΔの元bと$\overset{E}{\equiv}$等価なΣ項$t(a)$を表す木の部分木に対応するΣ項aの$\overset{E}{\equiv}$同値類$[a]$からなる．$\sigma \in \Sigma_{s_1 \cdots s_n, s}$に対する演算$\sigma_U : U_{s_1} \times \cdots \times U_{s_n} \to U_s$は，全域的な演算$\sigma_{T[\Sigma,E]} : (T_{\Sigma,s_1}/\overset{E}{\equiv}_{s_1}) \times \cdots \times (T_{\Sigma,s_n}/\overset{E}{\equiv}_{s_n}) \to (T_{\Sigma,s}/\overset{E}{\equiv}_s)$の定義域$U_{s_1} \times \cdots \times U_{s_n}$，値域$U_s$への制限であり必らずしも全域的ではない．

このUに関して次の命題が成り立つ．

命題3.1　任意の仕様記述(Σ, E, Δ)に対して，$U[\Sigma, E, \Delta]$は$\mathcal{O}\text{-}\mathcal{A}lg_\Sigma[E, \Delta]$の始代数である．

さて，このように定められた不完全指定形抽象データタイプの仕様記述と従来の完全指定形の仕様記述との違いを，典型的な抽象データタイプであるStackを例にとって見てみよう．ここで考えるStackの演算はNEWSTACK, PUSH, POP, TOP, ISNEWであり，それらの意味は[1~4]等で見られるものと同じとする．ただし，NEWSTACKに対して，POPおよびTOPは未定義であるとする．

次ページの図3.1および図3.2は，それぞれ，不完全指定形および完全指定形の立場からのStackの仕様記述である．尚，完全指定形の仕様記述では，未定義部分のある演算を取り扱うために特別な値UNDEFINEDを用いている．

```
type Stack[Element]
syntax (Σ)
    NEWSTACK :  → Stack
    PUSH : Stack × Element → Stack
    POP : Stack → Stack
    TOP : Stack → Element
    ISNEW : Stack → Bool
semantics (E)
    (1) ISNEW(NEWSTACK) = TRUE
    (2) ISNEW(PUSH(s,e)) = FALSE
    (3) POP(PUSH(s,e)) = s
    (4) TOP(PUSH(s,e)) = e
defined terms (Δ)
    φ
```

図3.1 Stackの不完全指定形仕様記述

```
type Stack[Element]
syntax
    NEWSTACK :  → Stack
    PUSH : Stack × Element → Stack
    POP : Stack → Stack
    TOP : Stack → Element
    ISNEW : Stack → Bool
semantics
    (1) ISNEW(NEWSTACK) = TRUE
    (2) ISNEW(PUSH(s,e)) = FALSE
    (3) POP(PUSH(s,e)) = s
    (4) POP(NEWSTACK) = UNDEFINED
    (5) TOP(PUSH(s,e)) = e
    (6) TOP(NEWSTACK) = UNDEFINED
```

図3.2 Stackの完全指定形仕様記述

不完全指定形と完全指定形の仕様記述の最も大きな差は等式集合にみられる。例えばこの例では、完全指定形の仕様記述の等式集合から演算の未定義性に関わる等式を除いたものが不完全指定形仕様記述の等式集合になっている。

さらに厳密に言えば、完全指定形仕様記述においてはすべての等式を、ok-propagation, error-propagation [1] の考え方に従って変形し、Stackの本質とは関係しないような余分の等式を加えなければならない。これに対して、不完全指定形の仕様記述の等式集合はStackの本質を表すと思われる等式のみからなり簡潔である。

もう一つの差は、不完全指定形の場合にはΔの指定が必要であるのに対して完全指定形ではそれを必要としない。従っ

てこの点に関しては完全指定形の方が簡潔であるといえる．しかし，実際の抽象データタイプの多くは，Stackの例でもそうであるように，どのデータ対象からも適当な合成演算によってある定数に到達できるという意味での連結性を持っているので，$\Delta = \Phi$であることが多い．また，ΦでなくてもΔの表現は多くの場合簡単である．

以上のように，不完全指定形の仕様記述は，従来の完全指定形の仕様記述と形式的にも，また，その意味のとらえ方においても基本的には同じで，しかも，演算の未定義性の取り扱いの煩雑さを軽減している．しかしながら次のような問題点がある．上のStackの場合には使用されなかったが，一般にはIF THEN ELSE演算の入った等式がよく用いられる．IF THEN ELSE (t, a, b)はt = TRUEのときはa，t = FALSEのときはbとなる演算であり，IF THEN ELSEと他の計算関数との合成は，

$$\text{IF THEN ELSE}(f(x_1,\cdots,x_\ell), g(y_1,\cdots,y_m), h(z_1,\cdots,z_n))$$

$$= \begin{cases} g(y_1,\cdots,y_m) ; & f,g,h\text{が定義されかつ } f(x_1,\cdots,x_\ell) = \text{TRUE} \\ h(z_1,\cdots,z_n) ; & f,g,h\text{が定義されかつ } f(x_1,\cdots,x_\ell) = \text{FALSE} \\ \text{未定義} ; & \text{その他} \end{cases}$$

で定義される．この定義の考え方は，プログラムの不動点理論や完全指定形仕様記述に見られる，未定義要素$\perp$やERROR要素による未定義性の取扱いに必要な拡張されたIF THEN ELSE

の定義の考え方とは異なる．従って，IF THEN ELSEの入った等式から直観的に意味を捉える際にその等式をプログラムとみなして理解すると仕様記述の意味を誤って理解する恐れがある．不完全指定形の仕様記述の中でIF THEN ELSEを用いるときには引数の値が定義されるか，あるいは未定義であるかに細心の注意が必要である．

4　不完全指定形抽象データタイプの実現　不完全指定形抽象データタイプの実現は，従来の抽象データタイプの実現と同様に，derivorの概念を用いて定式化される．不完全指定形抽象データタイプの演算は部分関数なので，演算の未定義部分をどのように実現するかという点に注意が必要である．

本報告で定式化される実現（これは非負整数）は次のような考えに基づいている．今，仕様記述によって与えられた抽象データタイプA，B，Cがあり，Aの一部をBで，また，Bの一部をCで実現することを考える．このとき，Aの一部をBで実現するときに使われる情報はAおよびBの仕様記述だけであり，BがCによってどのように実現されているかという情報は使用されない（してはいけない）．さらには，BによるAの実現を考えるときには，Bの仕様記述から未定義になるようなB上の演算の組合せはわかっているので，定義されるど

のようなAの演算の組合せを決して未定義であるBの演算の組合せに対応しないようにする．このようにすれば，Bの演算の未定義部分がCによってどのように実現されてもAの演算の定義される部分には影響がないようにできる．換言すると，Aの演算の定義される部分だけに注目する限り，Bの演算をCの合成演算で実現するときにはBの演算の未定義部分を自由に実現してもかまわない．このことは，丁度，組合せ禁止（または don't care）のある論理関数の実現に自由度があることに類似している．

さて，以上の考え方に従って不完全指定形抽象データタイプの実現を定式化しよう．まず deriver の定義を与える．

定義4.1　Σ を S-sorted operator domain，Ω を S'-sorted operator domain とする．Σ から Ω への deriver d は，写像 $f: S \to S'$ と写像族 $\langle d_{w,s}: \Sigma_{w,s} \to (T_\Omega(Y))_{f(w),f(s)} \rangle_{w \in S^*, s \in S}$ との対である．ただし，f の定義域は自然に S^* へ拡張されるとし，$(T_\Omega(Y))_{f(w),f(s)}$ $(w = s_1 \cdots s_n)$ は，$y_i \in X_{f(s_i)}$ $(i = 1, \cdots, n)$ であるような sort $f(s)$ の $\Omega(Y)$ 項 $t(y_1, \cdots, y_n)$ すべての集合である．Bを Ω 代数とするとき，d-derived 代数 dB は，$(dB)_s = B_{f(s)}$ $(s \in S)$，かつ，各 $\sigma \in \Sigma_{w,s}$ に対して $\sigma_{dB} = (d_{w,s}(\sigma))_B$ で定義される Σ 代数である．

定義4.2　n を非負整数とする．仕様記述 (Σ, E, Δ) の n 実現は4項組 $(B, d, \equiv, h)$ である．ここに，Bは Ω 代数，d

```
type Arrayint
syntax
   NEWARRAY : ⟶ Array
   ASSIGN : Array × Integer × Element ⟶ Array
   ACCESS : Array × Integer ⟶ Element
   < , > : Array × Integer ⟶ Arrayint (pairing function)
   P1 : Arraint ⟶ Array (projection function)
   P2 : Arraint ⟶ Integer (projection function)
semantics
   (1) ACCESS(ASSIGN(a,i,e),i) = e
   (2) ASSIGN(ASSIGN(a,i,e),i',e')
       = IF (i=i') THEN ASSIGN(a,i',e')
                   ELSE ASSIGN(ASSIGN(a,i',e'),i,e)
   (3) P1(<a,i>) = a
   (4) P2(<a,i>) = i
   (5) <P1(b),P2(b)> = b
defined terms
   {ASSIGN(...(ASSIGN(NEWARRAY,i1,e1)...),in,en) | n = 1,2,...}
```

図4.1 Arrayintの仕様記述

はΣからΩへのderivor, $\equiv$はdB上のΣ合同, かつ, hは$T[\Sigma,E,\Delta]$からdBへのΩ単準同形写像である.

このΩ実現を直観的に理解するために, 配列と整数の直積の抽象データタイプArrayintによるStackのΩ準実現を例としてあげる. Stackは3章で与えられた仕様記述(Σ,E,Δ)(図3.1)で定められるΣ代数$T[\Sigma,E,\Delta]$とする. 抽象データタイプArrayintは図4.1の仕様記述で定められるものとする. ArrayintはS'-sorted Ω代数であるとする. さて, ΣからΩへのderivor dを図4.2のように定める. 次に, $dArrayint$上のΣ合同$\equiv$を次の集合を含

```
f-part
   Stack ↦ Arrayint
   Element ↦ Element
   Bool ↦ Bool
d-part
   NEWSTACK ↦ < NEWARRAY,0 >
   ISNEW ↦ "P2(b)=0"
   PUSH ↦ <ASSIGN(P1(b),P2(b)+1,e),P2(b)+1)>
   POP ↦ <P1(b),P2(b)-1>
   TOP ↦ ACCESS(P1(b),P2(b))
```

図4.2 ΣからΩへのderivor d

も最小の二合同[†]とする.

$\{(\langle a,i\rangle,\langle a',i\rangle) \mid i=i' \text{ かつ } \forall j\leq i,\ ACCESS(a,j)=ACCESS(a',j)\}$

次に h は d と $\equiv$ から自然に定められる写像族とする. このとき h は1単準同形写像である. このことは, 少々面倒であるが d および $\equiv$ の定め方から示される. h が2単準同形写像ではないことは次のように示される. Stack の仕様記述から明らかなように, 深さ2の合成演算 PUSH(POP(a), e) が Stack 上では NEWSTACK に対して未定義であるが, dArrayint においては定義され $(PUSH(POP(NEWSTACK),e))_{dArrayint} \equiv NEWSTACK_{dArrayint}$ である. このことと, $h(NEWSTACK_{Stack}) = NEWSTACK_{dArrayint/\equiv}$ であることを合わせると $n=2$ の場合の条件 (h3) が満たされないことがわかる. 従って, (Arrayint, d, $\equiv$, h) は Stack の1実現である.

同じ Stack を従来の完全指定形の立場から実現するとほとんど同様の実現が得られる. しかし POP 演算は,

(*) IF P2(b)=0 THEN UNDEFINED ELSE ⟨P1(b),P2(b)-1⟩

という Arrayint 上の合成演算に対応付けられ, 上の1実現と比べると深さが大きくなっていて, 効率の点では1実現の方が優れている. また, Stack を使って他の抽象データタイプを完全指定形の立場で実現するとき, UNDEFINED を故意に使う場合は例外として, POP 演算の前には通常現在のスタックが空か否

---

† 関係 R を含む最小の二合同は, 一般には存在しない.

かをチェックする必要があるので，そのための IF THEN ELSE と上の(*)の IF THEN ELSE がむだな重複となる．ところが，不完全指定形抽象データタイプの N 実現では d と三をうまく工夫することによってそのような重複を避けることができる．

完全指定形の場合の実現と不完全指定形での N 実現とのこのような違いは，Stack の例に限らず一般的に見られるものであり，A を B で実現し，B を C で実現し，C を D で実現し，… というように実現の段数が大きくなればなるほどその差は大きくなると予想される．

5 あとがき　本報告では，従来の完全指定形の立場からの抽象データタイプの仕様記述の手法における演算の未定義性の取扱いの煩雑さを軽減するために不完全指定形抽象データタイプを導入し，その仕様記述法を与えた．そしてこの仕様記述法によって得られる仕様記述は，完全指定形の立場からの仕様記述と比べて等式の数が少ないなどの利点があることを例でしめした．

さらに，不完全指定形の抽象データタイプの実現として，N 実現を定式化し，N 実現は従来の実現と比べて自由度が大きく，かつ，効率の点で優れていることを Arrayint による Stack の 1 実現の例でしめした．

今後検討すべき課題として，3 章で述べた IF THEN ELSE 演算

の取扱いに関する問題，また，それに関連して，N実現に recursion を許した場合に起こる問題，さらに，仕様記述やN実現の正当性の証明に関する問題などが考えられる．

末筆ながら，御指導賜わる名古屋大学福村晃夫教授，並びに，討論して頂いた研究室の諸氏に深謝する．

文　献

[1] J.A.Goguen, J.W.Thatcher and E.G.Wagner; An initial algebra approach to the specification, correctness and implementation of abstract data types, IBM Research Report, RC-6487 (1976).

[2] J.V.Guttag, E.Horowitz and D.R.Musser; Abstract data types and software validation, CACM,21,12(1978).

[3] 嵩，谷口，杉山，萩原，鈴木，奥井；プログラムの仕様記述の代数的手法について，信学技報 AL78-5 (1978).

[4] B.H.Liskov and S.N.Zilles; Specification techniques for data abstraction, IEEE Trans. on SE, SE-1, 1(1975).

[5] 杉山，谷口，嵩；代数的仕様記述に基づくプログラム設計の一例，信学技報 AL79-13 (1979).

[6] 鈴木，杉山，萩原，谷口，嵩，奥井；プログラムの仕様と実現化の代数的記述，信学技報 AL78-46 (1978).

[7] J.W.Thatcher, E.G.Wagner and J.B.Wright; Data type specification: Parameterization and the power of specification techniques, Proceedings SIGACT 10th Annual Symposium on Theory of Computing.

```
type Array[Integer,Element]
syntax ( )
   NEWARRAY : Array
   ASSIGN : Array   Integer   Element   Array
   ACCESS : Array   Integer   Element
semantics ( )
   (1) ACCESS(ASSIGN(a,i,e),i) = e
   (2) ASSIGN(ASSIGN(a,i,e),i',e')
       = IF (i=i') THEN ASSIGN(a,i',e')
                   ELSE ASSIGN(ASSIGN(a,i',e'),i,e)
defined terms ( )
    ASSIGN(ASSIGN(...(ASSIGN(NEWARRAY,i1,e1),i2,e2)...),in,en)
    n = 1,2,...
```