

モノドに基づく代数仕様の書換え

梅村晃広

NTT 基礎研究所

本稿では、代数仕様についてのモノド解釈という新しい意味論を定義し、これに基づく仕様書換え方法を示す。Moggi は、値から値への関数と値から計算への関数との関係がモノドによって与えられることを示した。本稿では、これを代数仕様に応用し、通常解釈の拡張としてのモノド解釈を定義する。そして、この意味論に基づいた、仕様書き換えのためのモディファイアの定義方法を提案する。本稿の方法はエラー処理の追加に限らず、さまざまな書換えに対応し得るものである。

Modification of algebraic specifications
based on monads

Akihiro Umemura

NTT Basic Research Laboratories

3-9-11, Midori-cho, Musashino-shi, Tokyo 180 Japan

akihiro@wink.ntt.jp

In this paper, we show the formal semantics of modification of algebraic specifications. Moggi showed the monadic structure of the relation between functions from values to values and functions from values to computations. We use this relation to define monad interpretation of specification, which is the extension of usual interpretation. After that, we show how to define syntactic modifiers based on monads. Throughout this paper, we use an example of extending a specification with error handling, but this method is not restricted to error handling.

1 はじめに

代数仕様においては、例えばスタックの仕様は図1(a)のように与えられる。このように、仕様はソートの集合 (sort で始まる行)、オペレータの集合 (op で始まる行)、そして等式の集合 (eq で始まる行) からなる。代数仕様に対する意味論は次のように与えられる: (i) まず、各ソートに集合を、各オペレータに関数を対応付ける「代数」が与えられる; (ii) この関数が等式の集合を満たすとき、その代数は仕様のモデルとなる; (iii) このモデルを集めたもの (あるいは、モデルのうちの一つ) がその仕様の意味論となる。

さて図1(a)の仕様を、スタックが空 (nil) のときに pop をするとエラーになるように変形することを考える。このとき、ただ

eq pop(nil) = error

を加えたのではうまくいかない。pop の返す値は Stack でなければならないからである。そこで、図1(b)のようにすることが考えられる。

本稿では、基本仕様 (a)、pop と top がエラーを起こし得ること、そして eq pop(nil) = error という式、以上の情報から (b) のようにエラーを考慮した仕様を導き出す書換えにともない意味がどのように変化するかを、書換えについての意味論として与える。このことは、エラーモディファイアをあらかじめ定義し、用意しておくことを可能にする。エラーモディファイアは、(a) のような仕様から (b) のような仕様を作るための作用素である。

「仕様は最初から完べきに書けるものではない」というのはもっともらしい主張である。この主張のもとで機械によるサポートを考える場合には、仕様についての意味論だけでなく、仕様書換えについての意味論も考えるべきである。

本稿でのモナドの利用方法は、Moggi[5][6] や Wadler[8][7] をベースにしている。Moggi は非決定性や副作用などを持つ一般化されたラムダ計算の意味論を、モナドを使って与えた。Wadler はこれに関数型言語に応用し、副作用、入出力、非決定性、例外処理、コンティニューエーション操作などをきれいな形で関数型言語の中で扱う方法を示した。このように、モナドは非常に一般化された方法であり、その適用可能範囲は広い。代数仕様への応用を考えた場合にも、本稿で扱うエラー処理の追加の他にもさまざまな種類の仕様書換えについての統一的な意味論となりうる可能性を持っている。そこで本稿でも、なるべくモナドをエラー処理に限定するのを先延しにしながら議論を進める。

2節で従来の代数仕様による意味論を定義し、3節ではモナドを使った意味論と、2つの意味論のつながりを述べる。意味論と構文上の実際の仕様書換えとの関連を述べるのが4節である。ここまでのエラー処理に限定する前の部分である。ここまでの議論は、今後他の種類の書換えにも応用できる可能性を持つものである。5節で、話をエラー処理に限定して議論する。6節でまとめを行い、今後の課題を述べる。

2 代数仕様の基本概念

現在の代数仕様は、非常に複雑であるが、ここでは、モジュールパラメーターなどを除き非常に単純化して考える。

また、後の議論に便利のように若干標準的でない定式化をするが、本質的な変更は行っていない。
定義 2.1 S が有限集合で、 Ω が集合族 $\{\Omega_{w,s}\}_{w \in S^*, s \in S}$ のとき、組 (S, Ω) をシグニチャと呼ぶ。ここで S^* は S の要素の有限列の集合、つまり $S^* = \{s_1 s_2 \dots s_n \mid s_i \in S\}$ (S^* は空列も含む)。 S の要素をソートと言ひ、各 $\Omega_{w,s}$ の要素をオペレータという。オペレータ a が $\Omega_{w,s}$ の要素であるとき、 $a : w \rightarrow s$ と書く。

```

obj STACK is
  sorts Elt, Stack.
  op nil :-> Stack.
  op push: Elt Stack -> Stack.
  op pop : Stack -> Stack.
  op top : Stack -> Elt.
  var E : Elt.
  var S : Stack.
  eq top(push(E,S)) = E.
  eq pop(push(E,S)) = S.
endo.

```

(a) 基本仕様

```

obj STACK is
  sorts Elt, Stack, Elt?, Stack?.
  subsorts Elt < Elt?, Stack < Stack?.
  op error : -> Stack?.
  op nil :-> Stack.
  op push: Elt Stack -> Stack.
  op pop : Stack -> Stack?.
  op top : Stack -> Elt?.
  var E : Elt.
  var S : Stack.
  eq top(push(E,S)) = E.
  eq pop(push(E,S)) = S.
  eq pop(nil) = error.
endo.

```

(b) エラーを考慮した仕様

図 1: スタックの仕様

定義 2.2 シグニチャ $\Sigma = \langle S, \Omega \rangle$ と集合族 $V = \{V_s\}_{s \in S}$ があるとする。このとき、 $s \in S$ に対して s -項を次のように定義する。 $(t$ が s -項であることを $t : s$ と書く):

1. $v \in V_s$ ならば、 $v : s$.
2. $f : s_1, \dots, s_n \rightarrow s$ で $t_1 : s_1, \dots, t_n : s_n$ のとき、 $f(t_1, \dots, t_n) : s$.

特に V_s の要素を変数と呼ぶ。

論理式としては、代数仕様では通常、操作的な要求から、等式と条件付き等式のみを考える。

定義 2.3 シグニチャ Σ 上の論理式を定義する。

- あるソート s について、 t_1 と t_2 がどちらも s -項のとき、 $t_1 = t_2$ の形の式を等式と言う。
- e_1, e_2 が等式のとき、 $e_1 \Rightarrow e_2$ を条件付き等式という。
- 論理式は等式または条件付き等式である。

定義 2.4 仕様はシグニチャ Σ と論理式の集合 E の組 $\langle \Sigma, E \rangle$ である。

次に、 Σ -代数を定義する。記法が通常と若干異なるが、本質的には同一である。

定義 2.5 $\Sigma = \langle S, \Omega \rangle$ がシグニチャであるとする。 Σ -代数 A とは、 S の要素(ソート)に集合を、オペレータに関数を対応づける写像の組 $\langle A_{\text{sort}}, A_{\text{op}} \rangle$ である。ただし、 $f \in \Omega_{s_1, \dots, s_n, s}$ に対して関数 $A_{\text{op}}(f)$ の定義域及び値域は、 $A_{\text{op}}(f) : A_{\text{sort}}(s_1) \times \dots \times A_{\text{sort}}(s_n) \rightarrow A_{\text{sort}}(s)$ でなければならない。省略記法として、 $A_{\text{sort}}(s)$ を A_s 、 $A_{\text{op}}(f)$ を f_A と書く。 Σ を固定したときに、 Σ -代数の全体はカテゴリーを成す。このカテゴリーを $\Sigma\text{-Alg}$ とする。

次に、 Σ -代数による項の解釈を定義する。(本稿ではこれを標準解釈と呼ぶ。)

定義 2.6 Σ -代数 A に沿った環境 ρ とは、変数集合族 V の任意の変数 $v : s$ に対して、 $\rho[v] \in A_{\text{sort}}(s)$ を与える関数のことである。

Σ -代数 A による環境 ρ のもとの項 t の解釈 $A[t]_\rho$ を次のように定義する:

1. t が変数のとき $A[t]_\rho = \rho[t]$;
2. $A[f(t_1, \dots, t_n)]_\rho = (A_{\text{op}}(f))(A[t_1]_\rho, \dots, A[t_n]_\rho)$.

s -項 t について $A[[t]]\rho \in A_s$ とする。

Σ -代数がモデルであるかどうかの定義を次に述べる。

定義 2.7 論理式 e について $A \models_\rho e$ を定義する:

1. $A \models_\rho t_1 = t_2$ とは $A[[t_1]]\rho = A[[t_2]]\rho$ のことである;
2. $A \models_\rho e_1 \Rightarrow e_2$ とは $A \models_\rho e_1$ ではないか $A \models_\rho e_2$ であることである。

定義 2.8 論理式の集合 E に対して、 $A \models_\rho E$ は、すべての $e \in E$ に対して $A \models_\rho e$ となることである。 $A \models E$ は、すべての ρ に対して $A \models_\rho E$ となることである。仕様 $\langle \Sigma, E \rangle$ について $A \models \langle \Sigma, E \rangle$ は、 $A \models E$ のことである。このとき A は仕様 $\langle \Sigma, E \rangle$ のモデルであるという。仕様 $\langle \Sigma, E \rangle$ のモデル全体は $\Sigma\text{-Alg}$ の full subcategory とみなせる。

3 モナドによる解釈

集合のカテゴリ Set 上のモナドを考え、このモナドを使った項の解釈を考える。 Σ -代数の拡張概念として、モナド代数を定義し、モナド代数により項を解釈する^{†1}。このことは、Moggi[5]に従えば、各オペレータを「値」から「計算」への関数とみることに対応する。

定義 3.1 カテゴリ A 上のモナド (T, η, μ) は、関手 (functor) $T : A \rightarrow A$ 、自然変換 (natural transformation) $\eta : 1_A \rightarrow T$ 、自然変換 $\mu : T^2 \rightarrow T$ からなり、次を満たすものである。

$$\mu \circ \eta T = 1_T = \mu \circ T\eta, \quad \mu \circ \mu T = \mu \circ T\mu.$$

定義 3.2 カテゴリ A 上のモナド $T = (T, \eta, \mu)$ があるとき、 T による Kleisli カテゴリ A_T が次のように定義される。

- 対象 (object): A の対象を A_T の対象とする。
- 射 (arrow): A_T の射 $f : A \rightarrow B$ は、 A の射 $f : A \rightarrow T(B)$ である。
- 射の合成: f, g はそれぞれ A で $f : A \rightarrow T(B)$, A で $g : B \rightarrow T(C)$ とするとき、 $g * f = \mu(C) \cdot T(g) \cdot f$ 。
ここで $*$ が A_T での射の合成、 \cdot は A での射の合成である。

関手 $F_T : A \rightarrow A_T$ と $U_T : A_T \rightarrow A$ が次のように定義できる。 f, g を A の射 $f : A \rightarrow B$, $g : A \rightarrow T(B)$ とするとき、

$$F_T(A) = A, \quad F_T(f) = \eta(B) \cdot f, \quad U_T(A) = T(A), \quad U_T(g) = \mu(B) \cdot T(g)$$

関手 F_T と U_T は adjoint pair となる。

代数仕様では、積 (product) が使われるので、これについての条件が必要になる。そこで Moggi[5]の強モナドという概念を使う。

定義 3.3 Cartesian 閉カテゴリ A 上の強モナド (T, η, μ, τ) は次のものからなる。

- (T, η, μ) はモナド。
- 自然変換 $\tau(A, B) : A \times T(B) \rightarrow T(A \times B)$ は次式を満たすものである:
 - $T(\tau(A)) \cdot \tau(1, A) = \tau(T(A))$;
 - $T(\beta(A, B, C)) \cdot \tau(A \times B, C) = \tau(A, B \times C) \cdot (1_A \times \tau(B, C)) \cdot \beta(A, B, T(C))$;
 - $\tau(A, B) \cdot (1_A \times \eta(B)) = \eta(A \times B)$;

^{†1}本稿でのカテゴリ理論に関する記法はほぼ [4] にしたがうが、射の合成については 2 種類の中間記号 (\cdot と $*$) を使って、通常のカテゴリでの射の合成と Kleisli Category での射の合成を区別する。

$$\circ \tau(A, B) \cdot (1_A \times \mu(B)) = \mu(A \times B) \cdot T(\tau(A, B)) \cdot \tau(A, T(B)).$$

ここで、 τ と β はそれぞれ $\tau(A) : 1 \times A \rightarrow A$ 、 $\beta(A, B, C) : (A \times B) \times C \rightarrow A \times (B \times C)$ の自然同値である。

以下の定義をより簡潔に記すため、 α という一種の省略記法を導入する。 α は記法上はアリティが不定の自然変換のようなものである。

定義 3.4 Cartesian 閉カテゴリー \mathbf{A} 上の強モナド (T, η, μ, τ) があるとき、まず、自然変換 $\psi(A, B) : T(A) \times T(B) \rightarrow T(A \times B)$ を $\psi(A, B) = \mu(A \times B) \cdot T(\tau(A, B)) \cdot c(T(B), A) \cdot \tau(T(B), A) \cdot c(T(A), T(B))$ と定義する。ここで c は $c(A, B) : A \times B \rightarrow B \times A$ の自然同値である。このとき $\alpha(A_1, \dots, A_n) : T(A_1) \times \dots \times T(A_n) \rightarrow T(A_1 \times \dots \times A_n)$ をアリティ n についての帰納法で定義する;

- $n = 0$ のとき $\alpha() = \eta(1)$.
- $n = 1$ のとき $\alpha(A_1) = 1_{T(A_1)}$.
- $n = i + 1$ のとき $\alpha(A_1, A_2, \dots, A_{i+1}) = \omega(A_1, A_2, \dots, A_n) \cdot \psi(A_1, A_2 \times \dots \times A_{i+1}) \cdot (1_{T(A_1)} \times \alpha(A_2, \dots, A_{i+1}))$. ここで $\omega(A_1, A_2, \dots, A_n)$ は $A_1 \times (A_2 \times \dots \times A_n)$ と $A_1 \times A_2 \times \dots \times A_n$ の間の自然同値とする。

α の定義から、次式が成立する: $F_T(h_1 \times \dots \times h_n) = \alpha(A_1, \dots, A_n) \cdot (F_T(h_1) \times \dots \times F_T(h_n))$

以下では強モナドを (T, η, μ, α) と書くときがある。次に定義するのは、 Σ -代数の拡張概念であるモナド代数である。Set を、集合と関数のカテゴリーとする。

定義 3.5 $\Sigma = \langle S, \Omega \rangle$ をシグニチャ、 $\mathbf{T} = (T, \eta, \mu, \alpha)$ を Set 上の強モナドとすると、 Σ - \mathbf{T} -モナド代数 B は

1. $s \in S$ に対して集合 B_s を、
2. $f \in \Omega_{w,s}$ に対して関数 $f_B : B_{s_1} \times \dots \times B_{s_n} \rightarrow T(B_s)$ を、対応付けたものである。ただし $w = s_1 \dots s_n$ とする。

各モナド代数は、シグニチャを Kleisli カテゴリー Set_T と結びつけるものである。 Σ -代数の場合と同様に、 Σ - \mathbf{T} -モナド代数全体もカテゴリーを成す。

定義 3.6 シグニチャ $\Sigma = \langle S, \Omega \rangle$ 、強モナド $\mathbf{T} = (T, \eta, \mu, \alpha)$ について、 B, B' が Σ - \mathbf{T} -モナド代数とする。 Σ - \mathbf{T} -モナド代数準同形写像 $h : B \rightarrow B'$ は写像族 $h = \{h_s : B_s \rightarrow T(B'_s)\}_{s \in S}$ で、任意の $f \in \Omega_{w,s}$ について

$$\mu(B'_s) \cdot T(h_s) \cdot f_B = \mu(B'_s) \cdot T(f_{B'}) \cdot \alpha(B'_{s_1}, \dots, B'_{s_n}) \cdot (h_{s_1} \times \dots \times h_{s_n})$$

を満たすものとする。ただし $w = s_1 \dots s_n$ である。

Σ - \mathbf{T} -モナド代数のカテゴリー $\Sigma\text{-}\mathbf{T}\text{-MAlg}$ は、対象を Σ - \mathbf{T} -モナド代数とし、射を Σ - \mathbf{T} -モナド代数準同形写像とするカテゴリーである。identity arrow は $\eta_B = \{\eta(B_s)\}_{s \in S}$ で、射の合成は $h' * h = \{\mu(B'_s) \cdot T(h'_s) \cdot h_s\}_{s \in S}$ である。

次に、 Σ -代数の場合と同様に、 Σ - \mathbf{T} -モナド代数による項の解釈を与える。

その前に、先に定義 2.6 で定義した Σ -代数による項の解釈を、以降の議論で扱いやすいように若干変更する。定義 2.6 では s -項に対して A_s の要素を対応づけたが、これをカテゴリー理論的に、 s -項に対して 1 から A_s への関数を対応付けるように変更する。ここで 1 は Set の終対象 (terminal object), すなわち、1 要素から成る集合である。

定義 3.7 Σ -代数 A に沿った環境 ρ とは、変数集合族 V の任意の変数 $v : s$ に対して、 $\rho[v] : 1 \rightarrow A_s$ を与える関数である。

Σ -代数 A による環境 ρ のもとでの項 t の解釈 $A[t]\rho$ を次のように定義する:

1. t が変数のとき $A[t]\rho = \rho[t]$;
 2. $A[f(t_1, \dots, t_n)]\rho = f_A \cdot (A[t_1]\rho \times \dots \times A[t_n]\rho) \cdot e^n$.
- ただし $e^n: 1 \rightarrow 1^n$ は、 1 と 1^n との間の同形写像。この定義により、 s -項 t に対して $A[t]\rho: 1 \rightarrow A_s$ となる。

これをもとに、強モナドを使った項の解釈を定義する。

定義 3.8 強モナド $T = (T, \eta, \mu, \alpha)$ があるとき、 Σ - T -モナド代数 B に沿った環境 ρ とは、変数集合族 V の任意の変数 $v: s$ に対して、 $\rho[v]: 1 \rightarrow B_s$ を与える関数。

Σ - T -モナド代数 B による項 t のモナド解釈 $B[t]\rho$ を次のように定義する。

1. t が変数のとき:

$$B[t]\rho = \eta(B_s) \cdot \rho[t]: 1 \rightarrow T(B_s)$$

2. $t \equiv f(t_1, \dots, t_n)$ のとき:

$$B[t]\rho = \mu(B_s) \cdot T(f_B) \cdot \alpha(B_{s_1}, \dots, B_{s_n}) \cdot (B[t_1]\rho \times \dots \times B[t_n]\rho) \cdot e^n$$

以上の議論は、identity 強モナドを持ってきたときには、前節の Σ -代数、それに基づく解釈等と一致することを記しておく。

上述のようにして、仕様に対する 2 種類の解釈が定められた。 Σ -代数 A は $F_T \circ A$ により Σ - T -モナド代数に変換できる。ここで、 $\Sigma = \langle S, \Omega \rangle$ としたとき、 $F_T \circ A$ は次のように定義される: $s \in S$ に対して $(F_T \circ A)(s) = F_T(A_s)$; $f \in \Omega_{w,s}$ に対して $(F_T \circ A)(f) = F_T(f_A)$ 。

このことから、強モナド $T = (T, \eta, \mu, t)$ の $\eta(A)$ が各 A ごとに monomorphism ならば、 Σ -Alg は Σ - T -MAIg の中に埋め込める。 F_T を使った変換が、モデルであるという性質も保存する、というのが以下に述べる命題 3.9 である。

命題 3.9 各 $\eta(A)$ が monomorphism のとき、

$$A[t_1]\rho = A[t_2]\rho \text{ iff } (F_T \circ A)[t_1]\rho = (F_T \circ A)[t_2]\rho$$

上の命題において、各項の解釈は構成要素の意味をカテゴリー論的に合成したものである点に注意する。従って上の命題は自明ではないが、強モナドの性質から成立する。

4 モナド代数に基づく仕様の書換え

モナド解釈は標準解釈の文字通り「拡大解釈」であることを前節で示した。この解釈のもとでは例えば図 1(a) にそのまま

`eq pop(nil) = error`

を加えたものに対して意味付け可能である。とすると、図 1 の仕様 (a) から仕様 (b) へのシグニチャの変形は「どのオペレータを特に拡大解釈すべきか」の指定であると考えられる。この考え方を形式化する。

定義 4.1 シグニチャ $\Sigma = \langle S, \Omega \rangle$ のモナド指定 Υ とは Ω の部分集合族 $\Upsilon = \{\Upsilon_{w,s} \mid \Upsilon_{w,s} \subseteq \Omega_{w,s}\}$ である。仕様 (Σ, E) に対するモナド指定 Υ のもとでのモナド解釈 C を定義する。

1. $s \in S$ に対して集合 C_s を対応づける。
2. $f \in \Omega_{w,s} - \Upsilon_{w,s}$ に対して関数 $(fc)^{pre}: C_{s_1} \times \dots \times C_{s_n} \rightarrow C_s$ を考える。 $fc = \eta(C_s) \cdot (fc)^{pre}$ とする。

3. $f \in \Upsilon_{w,s}$ に対して関数 $f_C : C_{s_1} \times \dots \times C_{s_n} \rightarrow T(C_s)$ を対応づける。

これは明らかにモノド解釈の特殊な場合であり、 Σ -T-MAIlg のサブクラスの要素となる。

一方、ソート構成子 M があるとき、モノド指定 Υ のもとでの M によるシグニチャ $\Sigma = \langle S, \Omega \rangle$ のシグニチャ書換え $\sigma(M, \Sigma, \Upsilon)$ が次のように定義できる。シグニチャ書換えによって得られる仕様を $\langle S', \Omega' \rangle$ とすると、 Ω' は、 $f \in \Upsilon_{w,s}$ に対して $f \in \Omega'_{w, M(s)}$ 、また、それ以外の $f \in \Omega_{w,s}$ については $f \in \Omega'_{w,s}$ として得られる最少の集合族であり、 S' はこれに対応するソートの集合である。モノドの各自然変換に対応するオペレータを定め、Kleisli カテゴリーにおける射の合成をもとのカテゴリーでの合成で書き下す形で等式を变形してやれば、もとの仕様 $\langle \Sigma, E \rangle$ から、 $\sigma(M, \Sigma, \Upsilon)$ にいくつかのオペレータを加えたシグニチャを持つ仕様 $\langle \Sigma', E' \rangle$ を作るができる。このとき、 $\langle \Sigma, E \rangle$ のモデルとなるモノド解釈は $\langle \Sigma', E' \rangle$ の標準解釈のモデルと対応をとることができる。

5 エラー処理モノドによる代数仕様の書換え例

本節では、前節までの定義を用いて、エラー処理の場合の書換えを検討する。エラー処理に対応する Set 上の強モノド $EM = (T, \eta, \mu, \tau)$ は実は非常に trivial に定義できる。

- $T(A) = A \cup \{error\}$.
- $T(f) = \lambda x. \text{if } x = error \text{ then } error \text{ else } f(x)$.
- $\eta(A) = \lambda x. x$.
- $\mu(A) = \lambda x. x$.
- $\tau(A, B) = \lambda x. \text{if } \pi_2(x) = error \text{ then } error \text{ else } x$.

このとき (T, η, μ, τ) が強モノドになることがわかる。エラーモノドに対応する構文上のモノドモディファイアとして

$$EM = (? , eta, mu, tau, error)$$

を用意する。? は T に対応するソート構成子、eta, mu, tau は強モノドの自然変換に対応するもので、ここではソートについての polymorphic operator となる。error は値 *error* を各 $A \cup \{error\}$ に埋め込む関数に対応するもので、やはり polymorphic operator として扱う。

図 1(a) の基本仕様をエラーモノドのもとで拡大解釈するものとし、pop と top にモノド指定を行い、

$$\text{eq pop}(nil) = error$$

を加える。ここで error はエラーモノドモディファイアから提供される識別子である。すると図 2(a) のようになる(シンタクスは適宜定めた)。図中の Stack? はモノド指定を明示したものである。これを標準解釈で見ると、等式の部分を書き換える。これは Kleisli カテゴリーにおける射の合成 $f * g$ をもとのカテゴリーでの $\mu \cdot T(f) \cdot g$ で書き直したものに相当する。 $g = \eta \cdot g'$ のとき $f * g = f \cdot g'$ であることに注意。その結果、図 2(b) が得られる。図 1(b) と図 2(b) の違いは、図 2(b) では各 A に対する $A?$ の意味付けと error に対する意味付けが完全に決定している点である。

6 おわりに

本稿では、代数仕様についてのモノド解釈という新しい意味論を定義し、これに基づく仕様書換え方法を示した。さらに、モノド解釈にもとづくモノドモディファイアの定義方法を提案した。

```

obj STACK in EM interpretation is
  sorts Elt, Stack.
  op nil :-> Stack.
  op push: Elt Stack -> Stack.
  op pop : Stack -> Stack?.
  op top : Stack -> Elt?.
  var E : Elt.
  var S : Stack.
  eq top(push(E,S)) = E.
  eq pop(push(E,S)) = S.
  eq pop(nil) = error.
endo.

```

(a) モナド解釈記法

```

obj STACK with EM is
  sorts Elt, Stack.
  op nil :-> Stack.
  op push: Elt Stack -> Stack.
  op pop : Stack -> Stack?.
  op top : Stack -> Elt?.
  var E : Elt.
  var S : Stack.
  eq top(push(E,S)) = eta(E).
  eq pop(push(E,S)) = eta(S).
  eq pop(nil) = error.
endo.

```

(b) 標準解釈記法

図 2: エラーモディファイアで書き換えられた仕様

本稿で定義したモナド解釈は、Moggi の言い方を使えば、各オペレータに対して“値から計算への関数”を対応づけることを許した解釈である。エラー処理の場合は「ある値またはエラー」を計算と位置付けた。エラー処理に限らず、入出力や副作用などさまざまなものがモナドという枠組の中で計算と位置付けられることが Moggi 等によって示されている。今後、特に検討すべきことは、代数仕様の中で「状態」をモナドを用いて記述することである。この方向の検討は、オブジェクト指向言語に対する意味論にもつながると考えている。

参考文献

- [1] M. Barr and C. Wells; *Toposes, Triples and Theories*. Springer-Verlag, 1985.
- [2] J. A. Goguen and R. M. Burstall; The semantics of Clear, a specification language. *Lecture Notes in Computer Science*, Vol. 86, pp. 294-332, 1980.
- [3] Joseph Goguen, Timothy Winkler, Jose Meseguer, Kokichi Futatsugi, and Jean-Pierre Jouan-
naud; Introducing OBJ. *Technical Report SRI-CSL-92-03*, 1992. To appear in Goguen, J.A.,
editor, *Application of Algebraic Specification using OBJ*, Cambridge University Press, 1992.
- [4] J. Lambek and P. J. Scott; *Introduction to higher order categorical logic*. Cambridge University
Press, 1986.
- [5] Eugenio Moggi; Computational lambda-calculus and monads. In *5th IEEE Symposium on
Logic in Computer Science*, pp. 14-23, 1989.
- [6] Eugenio Moggi; Notions of computation and monads. *Information and Computation*, Vol. 93,
pp. 55-92, 1991.
- [7] Philip Wadler; Comprehending monads. In *Proceedings of the 1990 ACM Conference on Lisp
and Functional Programming*, pp. 61-78, June 1990.
- [8] Philip Wadler; The essence of functional programming. In *19th ACM Symposium on Principles
of Programming Languages*, pp. 1-14, Jan 1992.