

代数的仕様記述における詳細化
—特に抽象的順序機械の場合—

大阪大学基礎工学部 鈴木 一郎 杉山 裕二
谷口 健一 嵩 忠雄

基底代数を前提とする代数的仕様記述, 特に抽象的順序機械による関数の実現について, 一般的な理論的枠組みを与え, レコードの書込み, 読出しに時間のかかるディスクに関連した具体例を示す.

1. 代数的仕様

以下, 基本的な定義は(1), (2)等に従う. 仕様 \mathcal{D} は (S, Σ, ε) で与えられる. S, Σ, ε はそれぞれポート, 関数記号, 公理の集合である. 以下では, 各公理 $\in \varepsilon$ の左辺には同じ変数は複数回現れる, 右辺に現れる変数は必ず左辺に現れているものとする.

基底代数 B は仕様 $\text{spec}(B) = (S_B, \Sigma_B, \varepsilon_B)$ で表されるとする.⁽²⁾ $\text{spec}(B)$ には代数 B の集合の元に一対一に対応する定数関数 (B -定数と呼ぶ) があり, 公理 $\in \varepsilon_B$ は (一般には無限の)

関数の定義表として与えられる。また、二つのB-定数が等しいかどうかを判定するための関数“ \equiv ”が各ソートにある。仕様 $\mathcal{D} = (S, \Sigma, \varepsilon)$ が $\text{spec}(B)$ の部分仕様として含むとき、B-仕様であるという。どの二つの異なるB-定数 b_1, b_2 に対しても $b_1 \equiv b_2$ とはならないとき、 \mathcal{D} は無矛盾であるという (\equiv または $\equiv_{\mathcal{D}}$ は \mathcal{D} の公理 ε によって与えられる (Σ 上の項の集合 T_{Σ} 上の) 合同関係を表す)。

$$B\text{-仕様 } \mathcal{D} = (S_B + S, \Sigma_B + \Sigma_0 + \Sigma_R, \varepsilon_B + \varepsilon),$$

$$B'\text{-仕様 } \mathcal{D}' = (S_{B'} + S', \Sigma_{B'} + \Sigma'_0 + \Sigma'_R, \varepsilon_{B'} + \varepsilon')$$

がそれぞれ無矛盾であり、 $B \subseteq B'$ (B は B' の部分代数、同型は同一視)、 $\Sigma_0 \subseteq \Sigma'_0 \subseteq \Sigma_0 + \Sigma_R$ とする ($+$ は共通部分のない集合の和を表す)。値域が S_B のソートである $\Sigma_B + \Sigma_0$ 上の任意の項 t_1, t_2 に対し、条件

$$\lceil t_1 \equiv_{\mathcal{D}} t_2 \Rightarrow t_1 \equiv_{\mathcal{D}'} t_2 \rceil$$

が満たされるとき、 \mathcal{D}' は \mathcal{D} の (Σ_0 に着目した) 詳細化であるという。

この考えは、仕様が表すものをその始代数に限定せず、また、拡張 (enrichment) におけるように、すべての項の値がB-定数として定まることを要求しない。さらに、いわゆる符号化、複号化による同型を用いた考え⁽³⁾とも異なる。ここでは、対応自身を仕様 \mathcal{D}' に含めて考えている。この詳細化は推移律

が成立つ。同様の群組みで基底代数 Σ をパラメタ化できる。

2. 抽象的順序機械

無矛盾な β -仕様 $\mathcal{D} = (S_{\beta} + S, \Sigma_{\beta} + \Sigma, E_{\beta} + E)$ が次の条件 (i), (ii) を満たすとき、特に \mathcal{D} を抽象的順序機械と呼ぶ。⁽²⁾

(i) S は左にだけ一つのソートからなる。そのソートを state とする。

(ii) Σ の関数について、引数ソートの系列中にソート state は高々一回しか現れない。

さらに \mathcal{D} が条件

(iii) E の公理の左辺は $f(x_1, \dots)$ 又は $f'(g(x_1, \dots), y_1, \dots)$ の形をしている。ここで、 x_1, \dots, y_1, \dots は変数又は定数記号、 g はソートが state の関数記号。

(iv) $E_{\beta} + E$ の各公理の左辺が、 Σ の公理の左辺とも、又、自分自身の Σ の真部分項とも重なり得ない (いわゆる Church-Rosser の性質の十分条件)。

E を満たす場合は、

- ① 公理の形が状態遷移に直接対応しており、直感的にわかりやすい。
- ② 構造的帰納法を用いることが容易である。
- ③ \mathcal{D} が無矛盾であることが容易に示すことができる。

の性質がある。

表, ファシユタウシストア, "CPU", HDLC のプロトコル, ファイル管理システム等は抽象的順序機械として記述できる。

3. 抽象的順序機械による関数の実現

$M = (S_B + \{\text{state}\}, \Sigma_B + \Sigma_M, \varepsilon_B + \varepsilon_M)$ は抽象的順序機械とする。以下簡単のため, 値域がソート state である各 (状態遷移) 関数記号

$$g_j : \text{state}, a_{j_1}, a_{j_2}, \dots \rightarrow \text{state} \in \Sigma_M$$

に対し, " $g_j(\dots) = \dots$ " は了形の公理は ε_M に存在しないような M のみで考える。上の各 g_j に対し, 記号の集合

$$G_j = \{ g_j(b_{j_1}, b_{j_2}, \dots) \mid b_{j_1}, b_{j_2}, \dots \text{ はそれぞれソート } a_{j_1}, a_{j_2}, \dots \text{ の } B\text{-定数} \}$$

を考える。さらにすべての g_j に対するこれらの集合の和

$$\bigcup_{g_j \in \Sigma_M} \text{の状態遷移関数記号 } G_j$$

を考える。そのソート ε_P と書く。 P の元の乗列全体の集合を考える。そのソート ε_{P^*} と書く。連接 " \cdot " はすべての乗列について共通であるとする。 M の各非遷移関数記号

$$f_i : \text{state}, a_{i_1}, a_{i_2}, \dots \rightarrow a_i \in \Sigma_M$$

($a_{i_1}, a_{i_2}, \dots, a_i \in S_B$) に対し, P^* 上の関数記号

$$\tilde{f}_i : P^*, a_{i_1}, a_{i_2}, \dots \rightarrow a_i$$

を考へ、 Σ_M の各公理に現れるすべての

$$f_i(\alpha, \dots) \in \tilde{f}_i(\alpha, \dots) \quad \text{に,}$$

$$g_j(\alpha, \dots) \in g_j(\dots) \cdot \alpha \quad \dots (*) \quad \text{に}$$

それぞれ対応させておきかゝることにより得られるB'-仕様 \tilde{M} と書く(基底代数B'は、BにPやP', "·", head, tail を追加したものである)。

このとき、任意の

$$f_i : \text{state}, \alpha_{i_1}, \alpha_{i_2}, \dots \rightarrow \alpha_i$$

と、対応する

$$\tilde{f}_i : P^*, \alpha_{i_1}, \alpha_{i_2}, \dots \rightarrow \alpha_i$$

について、(*)の関係で対応するリ-ト state の任意の項 α と P^* の系列 α 、及び任意のB-定数 $b_{i_1}, b_{i_2}, \dots, b_i$ (それぞれリ-ト $\alpha_{i_1}, \alpha_{i_2}, \dots, \alpha_i$) に対し、

$$f_i(\alpha, b_{i_1}, b_{i_2}, \dots) \equiv_M b_i \iff \tilde{f}_i(\alpha, b_{i_1}, b_{i_2}, \dots) \equiv_{\tilde{M}} b_i$$

が成立する。以下では記法上の簡単のため、Mのかわりに \tilde{M} を用いて議論する。

Pを $P_I + P_E$ のように分ける。P_Iはユ-ザが直接コントロールできる状態遷移(例えば機械の一命令の実行、要求の受付)を表し、P_Eは受付けた要求に対する完了信号や割込み等の、システム内での"ε-move"を表す。

\tilde{M} に非遷移関数

$$M\text{-control} : P^* \rightarrow \text{bool}$$

がある。M-control は次の [条件 false. preserve] を満たさなければならない。

[条件 false. preserve]:

$$(i) M\text{-control}(\text{null}) \stackrel{\sim}{\equiv} \text{true} \quad (\text{null は空系列}), \text{ かつ}$$

$$(ii) M\text{-control}(p \cdot \alpha) \stackrel{\sim}{\equiv} \text{true} \Rightarrow M\text{-control}(\alpha) \stackrel{\sim}{\equiv} \text{true} \\ (p \in P, \alpha \in P^*).$$

M-control(α) は、初期状態から一連の状態遷移 α が機械として起りうるとき、かつそのときのみ真となるように述語である。具体的には、条件付ジャンプ命令の実行のあとでは、条件に従って決ったものしか実行されないので、依頼してはいないリクエストに対する完了信号が到来しないとかい、たこと記述する。

ある B_0 -仕様 \tilde{M}_0 について、集合 (ソート) P_0, P_0^* と二つの関数

$$f_0 : P_0^*, \lambda_{01}, \lambda_{02}, \dots \rightarrow \lambda_0$$

$$wd_0 : P_0^*, \lambda_{01}, \lambda_{02}, \dots \rightarrow \text{bool}$$

が与えられるとすると ($P_0, P_0^*, \lambda_{01}, \lambda_{02}, \dots, \lambda_0$ は B_0 -ソート)。wd₀

は次の性質

$$wd_0(p \cdot x) \stackrel{\sim}{\equiv} \text{true} \Rightarrow wd_0(x) \stackrel{\sim}{\equiv} \text{true} \quad (p \in P_0, x \in P_0^*)$$

をもつ。

ここで、次のような仕様 \mathcal{M} を考える (図1参照)。

- (1) \mathcal{M} は無矛盾.
 (2) \mathcal{M} に ν - ト P_0, P_0^* , 関数記号 f_0 がある.
 (3) \mathcal{M} は \tilde{M} の部分仕様として持つ.
 (4) \mathcal{M} に非遷移関数

$$\text{Interpret} : P_0^*, P^* \rightarrow \text{bool}$$

がある. Interpret は次の条件 (1) ~ (6) を満たす ($p \in P_0$, $x \in P_0^*$, $y, y_1 \in P^*$).

$$(1) \text{Interpret}(\text{null}, \text{null}) \stackrel{\mathcal{M}}{\equiv} \text{true}$$

$$(2) \text{Interpret}(x, y) \stackrel{\mathcal{M}}{\equiv} \text{true} \text{ かつ } \text{wd}_0(p \cdot x) \stackrel{M_0}{\equiv} \text{true} \\ \Rightarrow (\exists y' \in P^*) (\text{Interpret}(p \cdot x, y' \cdot y) \stackrel{\mathcal{M}}{\equiv} \text{true})$$

$$(3) \text{Interpret}(x, y) \stackrel{\mathcal{M}}{\equiv} \text{true}$$

$$\Rightarrow \text{wd}_0(x) \stackrel{M_0}{\equiv} \text{true} \text{ かつ } M\text{-control}(y) \stackrel{\tilde{M}}{\equiv} \text{true}$$

$$(4) \text{Interpret}(x, y) \stackrel{\mathcal{M}}{\equiv} \text{true} \text{ かつ } \text{Interpret}(x, y' \cdot y) \stackrel{\mathcal{M}}{\equiv} \text{true}$$

$$\Rightarrow y' \in P_E^*$$

$$(5) \text{Interpret}(x, y_1 \cdot y) \stackrel{\mathcal{M}}{\equiv} \text{true} \text{ かつ } y_1 \in P_E^*$$

$$\Rightarrow \text{Interpret}(x, y) \stackrel{\mathcal{M}}{\equiv} \text{true}$$

$$(6) \text{Interpret}(x, y_1 \cdot y) \stackrel{\mathcal{M}}{\equiv} \text{true}$$

$$\text{かつ } \exists \delta \in P_E \text{ かつ } M\text{-control}(\delta \cdot y) \stackrel{\tilde{M}}{\equiv} \text{true}$$

$$\Rightarrow (\exists y'' \in P^*) (\text{Interpret}(x, y'' \cdot \delta \cdot y) \stackrel{\mathcal{M}}{\equiv} \text{true}).$$

Interpret(α , β) は, 上のレベルの系列 $\alpha \in P_0^*$ に対し, 順序機械 \tilde{M} で状態遷移 $\beta \in P^*$ が可能であり, しかも α に対し

後起りする \tilde{M} の遷移は P_E で表される "ε-move" のみである (条件 (4)) の否かを示す述語であり, 機械 \tilde{M} の可能な状態遷移 (M-control) とさらに制限する (条件 (3)) "プログラム" と表すと考えられる. 条件 (1), (2) は上のレベルで意味のある (wd_0 が真) 系列に対して \tilde{M} で必ず "実行" できることを示す. 条件 (5), (6) は P_E の "ε-move" とプログラムでコントロールできないことを表す.

f_0 に対し, \tilde{M} に関数

$$f : P^*, A_{01}, A_{02}, \dots \rightarrow A_0$$

があり, 任意の B-定数 b_{01}, b_{02}, \dots (それぞれ A_{01}, A_{02}, \dots) と B-定数 $\alpha \in P_0^*, \beta \in P^*$ について,

$$\text{Interpret}(\alpha, \beta) \equiv_{\mathcal{M}} \text{true}$$

$$\Rightarrow f_0(\alpha, b_{01}, b_{02}, \dots) \equiv_{\mathcal{M}} f(\beta, b_{01}, b_{02}, \dots)$$

が成立するとき, 「 f_0 は wd_0 のもとで, \mathcal{M} において \tilde{M} により定義される」という. さらに \mathcal{M} が (f_0 に着目した) \tilde{M}_0 の詳細化になっているとき, 「 \tilde{M}_0 の f_0 は wd_0 のもとで, \mathcal{M} において \tilde{M} により実現される」という.

Interpret による対応の概念は, "derivator" による詳細化や同型写像を用いる方法よりも一般的で, "ε-move" を含む場合のように, これらの枠組みが必ずしも適当とは思われない例についても, 自然な記述が行なえる.

4. 例

各基底代数がもつソート(集合)を図2に示す。Spec Table(図3)は抽象的な“表”を表す。表の検索項目となる値の集合が address, 表に書かれる値の集合が record である。関数 access は、表に書かれた record の値を出力するが、指定された address の値で以前に record が書かれていなければ、特別な record の元, nullrecord をその値とする。Spec AbstractMemory(図4)は3.の \tilde{M}_0 に対応し, $P1$ が P_0 にあたる。関数 content は f_0 に対応し, 最後の GET(d) で指定された address d で表を検索した結果の record を値とする。ここでは content は全域関数であるので, wd_0 は省略されている。Spec Disksystem(図5)は3.の \tilde{M} にあたり, Table としてのディスクと, 書込み用バッファ(WriteBuffer), 読み出し用バッファ(ReadBuffer)を用いて, V コードの読み出し, 書込みに時間がかかる場合を記述している。P2 が3.の P にあたり, P2 の ω は $P_E = \{Complete\}$, 他は P_I である。書込み要求 $ReqW(d, r)$ によりレコード r が WriteBuffer に記憶され, 完了の ε -move Complete が起こるとディスクの d 番地の値として定義される。読み出しも同様であり, $ReqR(d)$ に対し Complete が起こると, ディスクの d 番地のレコードの値が, ReadBuffer の値として定義される。TJ は要求に対する仕事完了したかどうかを判定するための命令,

l : Busy フラグ (Busy) をセットせよ. もし Busy = true

ならば goto l , それ以外の場合は次の命令へ.

が 1 回実行されたことを示し, Complete による状態遷移 (Busy が false になる) を起こすまでくり返される.

Spec Definition By Disksystem (図 6) は 3. の \mathcal{M} にあてり,

① Abstract Memory, Definition By Disksystem 共に無矛盾.

② Interpret は 3. の条件 (1) ~ (6) を満たす.

③ 任意の $\alpha \in P1^*$, $\beta \in P2^*$ に対し,

$$\text{Interpret}(\alpha, \beta) \stackrel{\text{Definition By Disksystem}}{=} \text{true}$$

\Rightarrow (\exists B-定数 $r \in \text{record}$)

$$(\text{content}(\alpha) \stackrel{\text{Abstract Memory}}{=} r \text{ かつ}$$

$$\text{content}(\beta) \stackrel{\text{Definition By Disksystem}}{=} r).$$

が成立する (証明は略す). 従って, 3. の定義に従って,

「Abstract Memory の content は, Definition By Disksystem にあ

いて, Disksystem により実現される」

とある.

文献

- (1) 杉山, 谷口, 嵩: “代数的記述言語とその部分言語としての関数的プログラミンング言語”, 信学技報, AL 79-99 (1980-01).
- (2) —: “基底代数を前提とする代数的仕様記述”, 信学論 (採

録決定).

- (3) Ehrich and Lipect: "Proving Implementations Correct - Two Alternate Approaches", Information Processing 80.

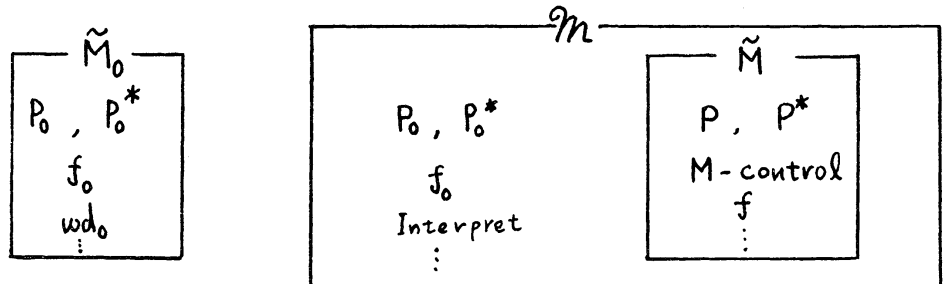


図1. \tilde{M}_0 と M の構造

Algebra0: address, record, bool,
 $P0 = \{PUT(d,r) \mid d \text{ is a B-constant of sort address, } r \text{ is a B-constant of sort record}\},$
 $P0^*$
 Algebra1: all sorts in Algebra0,
 $P1 = \{PUT(d,r) \mid d \text{ is a B-constant of sort address, } r \text{ is a B-constant of sort record}\}$
 $+ \{GET(d) \mid d \text{ is a B-constant of sort address}\},$
 $P1^*$
 Algebra2: all sorts in Algebra0,
 $type = \{ "WRITE", "READ" \},$
 $P2 = \{ReqW(d,r) \mid d \text{ is a B-constant of sort address, } r \text{ is a B-constant of sort record}\}$
 $+ \{ReqR(d) \mid d \text{ is a B-constant of sort address}\}$
 $+ \{Complete, TJ\},$
 $P2^*$
 Algebra3: all sorts in Algebra1, Algebra2

図2. LIST OF SORTS IN BASE ALGEBRAS

```
SPEC Table;

BASE Algebra0;
OP
  access : P0*, address -> record;
VAR
  x      SORT P0*;
  d,d'   SORT address;
  r      SORT record;
AX
  A1:   access(null,d) == nullrecord;
  A2:   access(PUT(d,r).x, d')
        == if d=d' then r else access(x,d');

END;
```

図3. Spec Table

```

SPEC AbstractMemory;

BASE Algebra1;
INCLUDE Table;
OP
  eraseGET : P1* -> P0*;
  content  : P1* -> record;
VAR
  x SORT P1*;
  d SORT address;
  r SORT record;
AX
  EG1: eraseGET(null) = null;
  EG2: eraseGET(PUT(d,r).x) = PUT(d,r).eraseGET(x);
  EG3: eraseGET(GET(d).x) = eraseGET(x);
  C1: content(null) = nullrecord;
  C2: content(PUT(d,r).x) = content(x);
  C3: content(GET(d).x) = access(eraseGET(x),d);

END;

```

4. Spec AbstractMemory

```

SPEC Disksystem;
BASE Algebra2;
INCLUDE Table;
OP
  ReadBuffer      : P2* -> record;
  WriteBuffer     : P2* -> record;
  Disk            : P2* -> P0*;
  RequestType     : P2* -> type;
  AddressRegister : P2* -> address;
  Busy            : P2* -> bool;

  M-control       : P2* -> bool;
  Testing         : P2* -> bool;
VAR
  x SORT P2*;
  d SORT address;
  r SORT record;
AX
  MC1: M-control(null) = true;
  MC2: FOR s = ReqW(d,r).x AND ReqR(d).x
        M-control(s) =
        == if M-control(x) then not Testing(x);
           else false;
  MC3: M-control(Complete.x)
        = if M-control(x) then Busy(x)
          else false;
  MC4: M-control(TJ.x) = M-control(x);

  I1: ReadBuffer(null) = nullrecord;
  I2: Disk(null) = null;
  I3: Busy(null) = false;
  I4: Testing(null) = false;

```

5. Spec Disksystem

```

RW: LET s = ReqW(d,r).x IN
ReadBuffer(s) == if M-control(s) then ReadBuffer(x),
WriteBuffer(s) == if M-control(s) then
    if not Busy(x) then r,
Disk(s) == if M-control(s) then Disk(x),
RequestType(s) == if M-control(s) then
    if not Busy(x) then "WRITE",
AddressRegister(s) == if M-control(s) then
    if not Busy(x) then d,
Busy(s) == if M-control(s) then true,
Testing(s) == if M-control(s) then Testing(x);

RR: LET s = ReqR(d).x IN
FOR G IN {ReadBuffer, WriteBuffer, Disk}
G(s) == if M-control(s) then G(x),
RequestType(s) == if M-control(s) then
    if not Busy(x) then "READ",
AddressRegister(s) == if M-control(s) then
    if not Busy(x) then d,
Busy(s) == if M-control(s) then true,
Testing(s) == if M-control(s) then Testing(x);

CPLT: LET s = Complete.x IN
ReadBuffer(s)
== if M-control(s) then
    if Busy(x) then
        if RequestType(x) = "READ" then
            access(Disk(x),AddressRegister(x))
        else ReadBuffer(x),
WriteBuffer(s) == if M-control(s) then WriteBuffer(x),
Disk(s)
== if M-control(s) then
    if Busy(x) then
        if RequestType(x) = "WRITE" then
            PUT(AddressRegister(x),WriteBuffer(x)).Disk(x)
        else Disk(x),
Busy(s) == if M-control(s) then false,
Testing(s) == if M-control(s) then Testing(x);

TJ: LET s = TJ.x IN
FOR G IN {ReadBuffer, WriteBuffer, Disk,
RequestType, AddressRegister}
G(s) == if M-control(s) then G(x),
FOR G IN {Busy, Testing}
G(s) == if M-control(s) then Busy(x);

```

END;

12) 5. (続 3)

SPEC Definition By Diskssystem;

```

BASE Algebra3;
INCLUDE Diskssystem;
OP
  content      : P1*      -> record;
  typical      : P1*      -> P2*;

  P-control    : P2*      -> bool;
  Requesting   : P2*      -> bool;
  Interpret    : P1*, P2* -> bool;
  Extract      : P2*      -> P1*;
VAR
  x SORT P1*;
  y SORT P2*;
  d SORT address;
  r SORT record;
AX
  CON1: content(x) == ReadBuffer(typical(x));

  TYP1: typical(null) == null;
  TYP2: typical(PUT(d,r).x) == Complete.ReqW(d,r).typical(x);
  TYP3: typical(GET(d).x) == Complete.ReqW(d).typical(x);

  PC1: P-control(null) == true;
  PC2: FOR s = ReqW(d,r).y AND ReqR(d).y
        P-control(s)
        == if P-control(y) then
            not Busy(y) and not Testing(y) and not Requesting(y)
            else false;
  PC3: P-control(Complete.y)
        == if P-control(y) then Busy(y)
            else false;
  PC4: P-control(TJ.y)
        == if P-control(y) then
            Busy(y) or Testing(y) or Requesting(y)
            else false;

  R1: Requesting(null) == false;
  R2: FOR s = ReqW(d,r).y AND ReqR(d).y
        Requesting(s) == if M-control(s) then true;
  R3: Requesting(Complete.y)
        == if M-control(Complete.y) then Requesting(y);
  R4: Requesting(TJ.y) == if M-control(TJ.y) then false;

  IPT1: Interpret(x,y)
        == if P-control(y) then
            x = Extract(y) and
            not Busy(y) and not Testing(y) and not Requesting(y)
            else false;

  EX1: Extract(null) == null;
  EX2: Extract(ReqW(d,r).y) == PUT(d,r).Extract(y);
  EX3: Extract(ReqR(d).y) == GET(d).Extract(y);
  EX4: FOR s = Complete.y AND TJ.y
        Extract(s) == Extract(y);
END;
```

6. Spec Definition By Diskssystem