代数的仕様記述における詳細化
一特に抽象的順序機械の場合一

大阪大学基礎工学部  鈴木一郎  杉山裕二
谷口健一  堺忠雄

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とれ、コード化し書き、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②は(\(S, \Sigma, \varepsilon\))で与えられる。\(S, \Sigma, \varepsilon\)とはそれぞれソート、関数記号、公理の集合である。以下では、合公理\(e\)の左辺には同じ変数は複数回現れず、右辺に現れた変数は必ず左辺に現れていないものとする。

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\(B\)の集合の元に一对一に対応する定数関数(\(B-\)定数と呼ぶ)があり、公理\(e\)\(B\)は(一般には無限の)
関数の定義表として示される。ここで、互いのB-定数が等しいかどうかを判定するための関数"="が各ソートにある。仕様\( \mathcal{D} = (S, \Sigma, \mathcal{E}) \)が\( \text{spec}(B) \)を部分仕様として含むとき、B-仕様であるという。どの二つも異なるB-定数\( b_1, b_2 \)に対しても\( b_1 \equiv b_2 \)にならないとき、\( \mathcal{D} \)は無矛盾であるという（\( \equiv \)は\( \mathcal{D} \)の公理をによって与えられる（\( E \)の頂の集合\( T_E \)上の）合同関係を表す）。

B-仕様\( \mathcal{D} = (S_B \times S, \Sigma_B \times \Sigma_0 + \Sigma_R, \mathcal{E}_B + \mathcal{E}) \)

B'-仕様\( \mathcal{D}' = (S_B' \times S', \Sigma_B' \times \Sigma_0' + \Sigma_R', \mathcal{E}_B' + \mathcal{E}') \)

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）、\( \Sigma_0 \subseteq \Sigma_0 ' \subseteq \Sigma_0 + \Sigma_R \)となる（\( \Sigma \)は共通部分、\( \Sigma_R \)集合の絶を表す）。値域が\( S_B \)のソートである\( S_B + \Sigma_0 \)上の任意の頂\( t_1, t_2 \)に対して、条件

\[
\text{「} t_1 \equiv t_2 \Rightarrow t_1 \equiv t_2 \text{」}
\]

が満たされるととき、\( \mathcal{D} \)は\( \mathcal{D}' \)の（\( \Sigma_0 \)に着目した）詳細化であるという。

この考えでは、仕様が表すものをその始代数に限定せずに、また、拡張（enrichment）におけるように、すべての頂の値が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、対応自身も仕様\( \mathcal{D}' \)に含めて考えている。この詳細化は推移律
が成立つ。同様に極座標で基底代数をパラメータ化できる。

2. 抽象的順序機械

無矛盾なB-仕様 \( \Sigma = ( S_B + S, \Sigma_B + \Sigma, \Sigma_B + \Sigma ) \) 必次の条件(i),
(ii)を満たすとき, 特に (i) 抽象的順序機械と呼ぶ。(2)

(i) \( S \) は有限数のソートからなる。 \( \forall \) ソート \( \text{state} \)

(ii) どの関数についても, 引数ソートの系列中にソート \( \text{state} \)

は高さ一回しか現れない。

さらに (iii) 式条件

(iii) 3 の公理の左辺は \( f(x_1, \ldots) \) 又は \( f'(g(x_1, \ldots), y_1, \ldots) \) の形

をしている。ここで, \( x_1, \ldots, y_1, \ldots \) 形変数又は定数記号,

\( g \) はソート \( \text{state} \) の関数記号。

(iv) \( \Sigma_B + \Sigma \) の各公理の左辺が, どの公理の左辺とも, 又, 自

分自身のど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）。

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

りやすい。

② 構造的帰納法を用いることが容易である。

③ \( \Sigma \) が無矛盾であることを容易に示すことができま

-3-
の性質がある。

表、プッシュダウンストア、「CPU」、HDLC・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される。

3. 抽象的順序機械による関数の実現

\( M = (S_B + \{\text{state}\}, \Sigma_B + \Sigma_M, \delta_B + \delta_M) \) を抽象的順序機械とす

る。以下簡単のため、値域がソート \( \text{state} \) である各（状態遷移）関数記号

\( g_j : \text{state}, a_1, a_2, \cdots \to \text{state} \in \Sigma_M \)

に対し、「\( g_j(\cdots) = \cdots \)」形の記号 \( \in \Sigma_M \) に存在しないよう

\( g_j \) を考えた。上の各 \( g_j \) に対し、記号の集合

\( G_j = \{ g_j(b_1, b_2, \cdots) : b_1, b_2, \cdots \in \text{state} \to \text{state}, a_1, a_2, \cdots \}

\cdot B-定数 \}

を考え、これらすべての \( g_j \) に対し \( G_j \) 小さな集合の和

\[ \bigcup_{g_j \in \Sigma_M} \text{状態変換関数記号} \quad G_j \]

を考え、これを \( \Sigma P \) と書き、\( \Sigma \) は \( \delta_B \) の全体の集合と

考え、これを \( \Sigma P^* \) と書き、連接 "*" は \( \Sigma \) に関する共通であると言った。\( M \) の各関数記号

\( f_i : \text{state}, a_1, a_2, \cdots \to a_i \in \Sigma_M \)

（\( a_i, a_2, \cdots, d_i \in S_B \)）に対し、\( P^* \) 上の関数記号

\( \tilde{f}_i : P^*, a_1, a_2, \cdots \to a_i \)
－参照し、$M$の各分理に現えると次の

\[ f_i(\lambda, \ldots) \Leftrightarrow \tilde{f}_i(\alpha, \ldots) \quad (\ast) \]

\[ g_j(\lambda, \ldots) \Leftrightarrow g_j(\ldots) \cdot \alpha \quad (\ast) \]

それぞれに対応させてみることにより得られる$B'-\text{仕様}$を

\[ \tilde{M} \quad \text{書く} \quad (\text{基底代数} B' \text{は、} B \text{に} P \text{や} P', \ldots, \text{head, tail を追加} \text{したものを}) \]

このとき、任意の

\[ f_i : \text{state} \cdot \lambda_i, \lambda_i_2, \ldots \rightarrow \delta_i \]

と、対応す

\[ \tilde{f}_i : P^*, \lambda_i, \lambda_i_2, \ldots \rightarrow \delta_i \]

について、$(\ast)$の関係に対応するリーレート state に任意の頂点 $\lambda$ と

$P^*$の列 $\lambda$, 及び任意の$B$-定数 $b_i, b_i_2, \ldots, b_i$ (これらはリーレート

$\lambda_i, \lambda_i_2, \ldots, \lambda_i$) に対し,

\[ f_i(\lambda, b_i, b_i_2, \ldots) \equiv \lambda_i \Leftrightarrow \tilde{f}_i(\alpha, b_i, b_i_2, \ldots) \equiv b_i \]

が成立する．以下に記法上の簡単のため、$M$のかわりに$\tilde{M}$

を用いて議論した．

$P$ は $P_1 + P_2$ のように分ける．$P_1$ はユーザが直接コントロールできる状態変換（例としては機械の一命令の実行、要求の受付け）を表し、$P_2$ は受付メディエーターに対応する完了信号や割込み等

の、システム内部の "move" を表す．

何に非変換関数
M-control : P* → 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-control(null) ≡ true (null は空字列), 02
(ii) M-control(p•a) ≡ true ⇒ M-control(a) ≡ true
(p ∈ P, a ∈ P*).

M-control(a) は、初期状態から一連の状態遷移 a が起こり、進化するとき、a と a と a と a と a と a と a と a と a と a を繰り返すような述語である。具体的には、条件付きジャンプ命令の実行の考えに行われる、条件に従うか決まったものしか実行されないという、依存していないリクエストに対応した信号が到達しないといけない、ことに記述する。

ある Bo は、集合 (セット) P0, P0* を関数

f0 : P0*, A01, A02, ... → A0

wd0 : P0*, A01, A02, ... → bool

が与えられて、M0(p0, P0*, A01, A02, ..., A0 は Bo で)。wd0 が次のように

wd0(p•x) ≡ true ⇒ wd0(x) ≡ true (p ∈ P0, x ∈ P0*)

を示す。

ここで、次のようには状態 M0 を考えた (図 1 参照)。
(1) \( M \) は無矛盾。
(2) \( M \) にリーチ \( P_0, P_0^* \), 閉数記号 \( f_0 \) が含まれる。
(3) \( M \) は \( \tilde{M} \) で部分仕様としてもよい。
(4) \( M \) に非逆移閉数

\[
\text{Interpret} : P_0^*, P^* \rightarrow \text{bool}
\]
がある。\( \text{Interpret} \) は次の条件 (1) - (4) が満たされる (\( p \in P_0, x \in P_0^*, y, y_1 \in P^* \))。

(1) \( \text{Interpret}(\text{null}, \text{null}) \equiv \text{true} \)

(2) \( \text{Interpret}(x, y) \equiv \text{true} \) かつ \( w_{d_0}(p \cdot x) \equiv \text{true} \)

\( \Rightarrow (\exists y' \in P^*)(\text{Interpret}(p \cdot x, y' \cdot y) \equiv \text{true}) \)

(3) \( \text{Interpret}(x, y) \equiv \text{true} \)

\( \Rightarrow w_{d_0}(x) \equiv \text{true} \) かつ \( \text{M-control}(y) \equiv \text{true} \)

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

\( \Rightarrow y' \in P_E^* \)

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

\( \Rightarrow \text{Interpret}(x, y) \equiv \text{true} \)

(6) \( \text{Interpret}(x, y' \cdot y) \equiv \text{true} \)

\( \Rightarrow y' \in P_E \) かつ \( \text{M-control}(y \cdot y) \equiv \text{true} \)

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

\( \text{Interpret}(x, y) \) は、上のレベルの系 \( x \in P_0^* \) に対して、順序機構 \( \tilde{M} \) で状態変移 \( y \in P^* \) が可能であり、しかし \( x \) に対しても

-7-
接起るうる$\tilde{M}$・駆動は$P_E \vdash$ 表される "$\varepsilon$-move" のみである（条件(4)) が否かを示す延髄であり、機能$\tilde{M}$の可能な状態駆動（M-control）をさらに制限する（条件(1)） "プロブノウム" を表すと考えられる。条件(1), (2)は上のレベルで意味のある ($w_0$ が真) 制限において$\tilde{M}$ は必ず "実行" されていることを示す。条件(5), (6)は $P_E \vdash "\varepsilon$-move" でプロブノウム2 コントロールで
きないことを表す。

$f_0$ に対し、$\tilde{M}$ に関して

$$ f : P^* \times A_{01} \times A_{02} \times \cdots \to A_0 $$

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

Interpret ($x, \beta$) $\equiv$ true

$$ \Rightarrow f_0 (x, b_{01}, b_{02}, \cdots) \equiv f_0 (\beta, b_{01}, b_{02}, \cdots) $$

が成立するとき、「$f_0$ は $w_0$ のもとで, $\varrho_m$ において$\tilde{M}$ により
定義される」という。さらに $\varrho_m$ 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 $w_0$ のもとで、$\varrho_m$ によ
って$\tilde{M}$ により実現される" という。

Interpret による対応の概念は、"derivor" による詳細化や同型写
像を用いる方法よりも一般的で、"$\varepsilon$-move" を含んだ場合のよ
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. 割

各具体数がもつソート（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・集合がaddress、表に書かれ値・集合がrecordである。関数accessは、表に書かれたrecordの値を出力すガ、指定したaddressの値が以前にrecordが書かれているければ、特別なrecordの元、nullrecordをその値とす。Spec AbstractMemory（図4）は3."Poに対応し、P1がP0に対応し、最後のGET(d)で指定されたaddress dの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と、wd0は省略されている。Spec Disksystem（図5）は3."Moに対して、Tableとしてディスクと、書き込み用バッファ（WriteBuffer）、読み出し用バッファ（ReadBuffer）を用いて、コードが読み出し、書き込みに時間がかかる場合を記述している。P2が3."Oに対応し、P2の下にPE = Complete、他はP1である。書き込み要求RegW(d,r)によりコードとWriteBufferに記憶され、完了のe-move Completeが起こるとディスクリのd番地の値として定義される。読み出しも同様であり、RegR(d)に到りCompleteが起こるとディスクリのd番地のレコードの値が、ReadBufferの値として定義される。TJは要求に応じた仕事を完了したかどうかを判定したための命令、
l : Busyフラグ（Busy）をテストする。もしBusy = true
ならばgoto l で、そうでなければ次の命令へ。

が1回実行されることを示し、Completeによる状態遷移（Busy がfalseになる）が起こり、そこでくり返される。

Spec Definition By Disksystem は3.の9cにあり、

① AbstractMemory, 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α ∈ P1, β ∈ P2: に対し、

Interpret (α, β) ⇒ Definition By Disksystem true

⇒ (∃ B:定数 r ∈ record)

(content(α) ≡ Abstract Memory
content(β) ≡ Definition By Disksystem
が成立し(証明は略す)。従って3.の定義に従う。)

「Abstract Memoryのcontentは、Definition By DisksystemにおけるDisksystemにより実現される」

としよう。

文献


(2) ——：”基底代数と前提とす代数的性質記述”, 信学論（収
図 1. $\tilde{M}_0$ と $\tilde{M}$ の構造

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

-11-
SPBC AbstractMemory:

BASE Algebra;
INCLUDE Table;
OP
eraseGET : P1* -> P0*;
content : P1* -> record;
VAR
x SORT P1*;
d SORT address;
r SORT record;
AX
EQ1: eraseGET(null) == null;
EQ2: eraseGET(PUT(d,r).x) == PUT(d,r).eraseGET(x);
EQ3: 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

SPBC 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 = RegI(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
Fw: LET s = Req(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 = Req(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;

图5．(续1)
SPEC Definition By Disksystem;

BASE Algebra3;
INCLUDE Disksystem;
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.RegW(d,r).typical(x);
  TYP3: typical(GET(d).x) = Complete.RegW(d).typical(x);
  PC1: P-control(null) = true;
  PC2: FOR s = RegW(d,r).y AND RegR(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 = RegW(d,r).y AND RegR(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(RegW(d,r).y) = PUT(d,r).Extract(y);
  EX3: Extract(RegR(d).y) = GET(d).Extract(y);
  EX4: FOR s = Complete.y AND TJ.y
      Extract(s) = Extract(y);
END;

6. Spec Definition By Disksystem