論　文

あるスタイルに基づく順序機械型記述における詳細化の正しさの証明方法

正員　岡野　浩三[†]　　正員　東野　輝夫[†]　　正員　谷口　健一[†]

Proof Method for Correctness of Refinements of Algebraic Specification in Abstract Sequential Machine Style

Kozo OKANO[†], Teruo HIGASHINO[†] *and* Kenichi TANIGUCHI[†], *Members*

あらまし　代数的手法を用いた階層的設計法として，我々は要求仕様から実現プログラムまで同一の詳細化法に基づいて段階的に順次詳細化する方法を提案している．各レベルでは，そのレベルでの関数，処理の性質等の要求記述が行われ，そのレベルで閉じた記述になっている．上記の方法で順序機械型記述を行う際に，(イ)興味ある関数に対する要求のみ取り出す拡張射影を用いる方法，および，(ロ)通常のように前提条件を内部データに関する具体的な述語を用いて記述せず，外部入力の条件と実行制御上の条件で定義される到達正当性条件を用いて記述する方法等が有用であることが既にわかっている．本論文では，拡張射影あるいは，到達正当性条件を用いて記述された順序機械型記述間において，正しい詳細化であることを形式的に定義し，そうであることを示すための証明方法について新たに述べる．詳細化の自由度を大きな範囲で許しているにもかかわらず，これらの証明方法は従来の記述法に対する証明方法で必要とする手間や，考案すべき言明の量と比べ遜色がない．

キーワード　代数的仕様，階層的設計法，射影，順序機械型プログラム，プログラム検証法

1. まえがき

フォーマルアプローチによるプログラム開発法の一つに代数的手法[1),(3)~(5)]がある．代数的仕様は抽象的データ構造を設計，検証する立場から記述法や詳細化法に関して多くの研究がなされてきたが，これをプログラム開発に適用する立場からは，よい詳細化の枠組み，証明の方法論等の研究はまだ十分とは言えない．筆者らのグループでは我々の代数的言語[(7)]を定め，記述支援系，証明支援系，実行系からなるプログラム開発システム[(9)]を作成し，種々の研究を行ってきた．

これらの研究を通じて，順序機械型仕様記述において有用な概念として，拡張射影による記述方法，前提条件を陽に記述しない記述方法(以下，到達正当性条件による方法と呼ぶ)の二つを既に提案している[(8)]．これらの概念は以下に示す記述上の利点がある．

(1)　拡張射影は，等式集合を指定する従来の射影[(7)]の概念の拡張であり，要求を満たすような関数 F の入出力値の(対応に相当する)等式集合の集合を指定する．すなわち，補助的に用いるデータ構造や関数を仮に定め，それらを用いて本来計算したい関数 F の満たすべき性質を記述しておく．仕様としては F の入出力値さえその要求を満たすように実現すればよいとする．補助的な関数はそのまま実現する必要はない．拡張射影で，その要求を満たす F の入出力関係の全体を取り出す．このうちの任意の一つを満たすように実現すればよい．拡張射影を用いることにより，より自由度の高い要求仕様を意味上の厳密さを欠くことなしに記述することができる．

(2)　到達正当性条件による記述方法では，順序機械型プログラムの階層的設計での各レベルの記述において，各処理前の内部データが満たしているべき前提条件を陽に記述せず，そのレベルのプログラムの実行制御において初期状態から入力が正当である条件のもとで，実際に到達可能な状態であれば真となる述語 valid (到達正当性条件と呼ぶ)を定義し，その述語が成り立つことを前提条件にして処理の前後の関係を記述する

† 大阪大学基礎工学部情報工学科，豊中市
Faculty of Engineering Science, Osaka University, Toyonaka-shi, 560 Japan

スタイルをとる．このスタイルをとると，記述の際，前提条件をいちいち考慮しなければならない不都合や，条件が緩すぎる場合に実現に対する自由度を制限してしまう不都合，あるいはそのレベルのプログラム実行を考えると成立しない誤った条件を書いて記述全体が意味のないものとなるといった恐れがなくなる．

文献(8)では，記述上の利点を考慮にいれてこれらの概念を導入したが，これらを用いたテキスト(仕様)間の詳細化の正しさを，具体的にどのように示すかについては述べられていなかった．これらの実用的な証明法を与えることが必要である．そこで本論文では以下の2点について新たに述べる．

(A)　拡張射影を含む順序機械型プログラムの仕様記述t_1, t_2 において，t_2 が t_1 の正しい詳細化であることの十分条件および，その証明のための具体的な方法論を，本論文の**3.** で(拡張射影の形式的な定義を述べた後)述べる．証明方法として，t_1 で拡張射影を用いる際に補助的に用いた関数を t_2 で用いている関数で表すための対応関係を検証者が与えて，そのもとで従来の証明方法を適用する方法をとる．従って，拡張された記述クラスに対しても従来の証明方法が適用できる．

(B)　到達正当性条件による記述では，従来の合同関係の細分と言う概念では実用上妥当な詳細化を定義することはできない．そこで本論文の**4.** で，従来の定義の拡張を行う．また，正しい詳細化の証明方法を述べる．(記述の時点ではなく)証明の段階で valid が真なら成り立つような条件(言明)を検証者が与え，それをもとに(不変式であるという証明も含め)正しい詳細化の証明を行う方法をとる．この証明における論法，手間等は従来の前提条件を陽に記述するスタイルと比べほぼ同じである．

もちろん，テキストの記述に上記の二つの概念を共に用いている場合でもこれらの方法を適用できる．

2.　準備(設計法に関する諸概念)

ここでは，順序機械型記述による階層的設計法[8]と，その記述に用いる代数的言語[7]について簡単に述べる．

2.1　順序機械型記述による階層的設計法

我々は，プログラムを設計する際に，抽象的なレベルの記述から具体的なプログラムへ詳細化する方法をとる[8),(9)]．従って，各レベルに同一の言語を用い，処理の要求は定義された述語等を用いて意味定義の閉じた記述を行う．本論文ではこの記述クラスとして，抽象的順序機械型記述ASL/ASM(以下ASMと呼ぶ)[6]を選ぶ．

ASMでは状態を表すデータ型をもち，状態を引数にする関数として状態遷移関数と状態成分関数を定義する．ASMプログラムは，(イ)初期状態での各状態成分関数の値の記述(初期化)，(ロ)各状態遷移関数で各状態成分関数の値がどのように変わるか(状態遷移(内容)の定義)の記述，(ハ)一連の状態遷移の実行順序の指定記述(ループ関数の記述)，(ニ)初期状態から始まるどのような状態遷移後のどの成分値を出力するのかを指定する計算指定の記述，からなる．

ASMの階層的設計スタイルを以下に簡単に述べる．あるレベルで導入された状態遷移は，普通(ロ)の代わりに 状態遷移の性質の記述(ホ)がなされる．(ホ)では，その状態遷移関数を適用する前後の各状態成分関数値がどのような関係にあるべきかが述語の形式で記述される．ある状態遷移の各状態成分関数の記述に述語形式と定義形式が混在してもよい．このとき，この状態遷移の記述は(ホ)の記述とみなす．次のレベルでは，(複数の)より具体的な状態遷移関数を導入し，性質記述の対象となった状態遷移の一部または全部は，導入された状態遷移関数とループ関数を用いて，詳細化される．新たに導入された状態遷移関数については，(ホ)の記述がされる．性質記述を書くまでもない簡単な状態遷移については，(ホ)の代わりに状態遷移の定義(ロ)を記述する．残りの記述は上位の記述をすべて受け継ぐものとする．よって上位で用いられた状態成分関数は原則として，下位の記述でも用いられる．但し，拡張射影で用いられる補助的な状態成分関数は必ずしも下位で用いる必要はない．これは後述する．(ホ)がなくなるまで順次，詳細化を続ける．最下位の記述から(イ)，(ロ)，(ハ)，(ニ)を集めると実現プログラムとなる．

2.2　記述に用いる代数的言語

テキスト $t = (G, AX)$ は，表現式集合を指定するための開始記号をもたない文脈自由文法 $G = (V_N, V_T, P)$ と，公理集合 AX の対からなる．文脈自由文法 G において，非終端記号集合 V_N の任意の要素から生成規則の集合 P に従って生成される終端記号集合 V_T 上の記号列のすべての集合を，文法 G が指定する表現式集合 $E(G)$ と呼ぶ．記号 $p \in V_N$ は型を表す．

V_N 中の非終端記号 p に対して，V_N と V_T のいず

れにも属さない記号 x_p を複数設け，生成規則 $p \to x_p$ を考える．これらの生成規則を拡大生成規則 P' と呼び，拡大生成規則を G に追加して得られる文法を G' とする．G' の指定する表現式集合 $E(G')$ を項集合と呼び，その要素を項と呼ぶことにする．x_p は型 p の変数と考えられる．項 ξ の各変数に対して許される代入 ρ は変数 x_p にはその型(非終端記号)p から t 上で生成される表現式のみに限られるとする．テキスト t の公理集合 AX は，$\{\xi == \eta \mid \xi, \eta \in E(G')\}$ なる集合である．各 $\xi == \eta$ を公理と呼ぶ．

テキスト t は，表現式集合 $E(G)$ とその上の合同関係 $\equiv_t$ を指定する．

[定義 1] [合同関係 $\equiv_t$] 一つの公理 $\xi == \eta$ は ξ, η に対する任意の許される代入 ρ によって得られる表現式 $\rho(\xi)$ と $\rho(\eta)$ の等式集合を生成する．テキスト t の表現式集合 $E(G)$ 上の合同関係 $\equiv_t$ は，公理集合 AX 中の各公理が生成する等式集合を含む最小の合同関係とする[7]．

この合同関係は常に存在し一意に定まる[7]．他の言語では始代数等による意味論を採用しているが，ここでは合同関係による意味論を用いる．

テキストの V_N の部分集合 V_{NC} を指定し，それに属する非終端記号(型)M から生成される表現式を型 M の構成子表現式と言う．テキスト $t = (G, AX)$ および，表現式 $\xi \in E(G)$ に対して，表現式 ξ が，ある構成子表現式と $\equiv_t$ 上で合同関係にあるとき，表現式 ξ はテキスト t において値をもつ(値はその構成子表現式)と呼ぶことにする．以下，表現式集合と言うときなどは，構成子表現式が暗に指定されているものとする．

[定義 2] [合同関係を満たす関係] E_1, E_2 を $E_1 \subseteqq E_2$ なる表現式集合とし，$\equiv_1, \equiv_2$ をそれぞれその上の合同関係とする．条件「E_1 中の任意の表現式 ξ, η に対して，$\xi \equiv_1 \eta$ ならば，$\xi \equiv_2 \eta$ が成り立つ」が成立するとき，合同関係 $\equiv_2$ は，合同関係 $\equiv_1$ を満たすと言い，$\equiv_1 \subseteqq \equiv_2$ (あるいは $\equiv_2 \supseteqq \equiv_1$)と表現する．

プログラムの仕様を書くためには，どの項の計算を実現するかという(興味ある項の)概念が必要になる．

τ を実現して欲しい関数を表す(変数付きの)項の集合とする．表現式集合とその上の合同関係 $\equiv$ に対し，表現式集合中の相異なる二つの構成子表現式間に合同関係 $\equiv$ が成り立たないとき，合同関係 $\equiv$ は無矛盾であると言う．

[定義 3] [正しい実現] 二つのテキスト $t_1 = (G_1, AX_1), t_2 = (G_2, AX_2)$ において，$E(G_1) \subseteqq E(G_2)$ とする．$\tau \subseteqq E(G_1')$ を満たす項集合 τ に関して，次の条件が成立するとき，τ に関してテキスト t_2 はテキスト t_1 の正しい実現であると言う．

(1) $\equiv_{t_1} \subseteqq \equiv_{t_2}$ が成り立つ．

(2) τ 中の各項 ξ に対し t_1 上で許される任意の代入を行って得られた表現式が t_2 において値をもつ．

(3) $\equiv_{t_2}$ は無矛盾である．

直観的には，ある項 $\xi \in \tau$ に関して正しい実現をすれば，その正しい実現をしたテキスト上で合同関係により項 ξ の値を求めることができる．

2.3 射　　影

射影[7]を用いることによって，参照したテキスト s の合同関係から興味ある等式集合のみ取り出せる．このような枠組みは，他の手法ではhidden 関数等の概念が用いられている[2]．我々の言語では，合同関係で同等の概念を定義する．これにより，簡明な意味定義が可能となる．

[定義 4] [射影] テキスト t 中の射影文 $\xi == \eta$@**text** s は，テキスト t 上で ξ, η に対して許される代入 ρ のうち，$\rho(\xi) \equiv_s \rho(\eta)$ を満たす表現式の対 $\langle \rho(\xi), \rho(\eta) \rangle$ の集合を等式集合として生成する．但し，s 中に 射影文はないとする．テキスト t の定める合同関係 $\equiv_t$ は各公理，射影文が生成する等式集合を含む最小の合同関係である†．

例えば整数を逐次入力し，過去の入力系列中に現在の入力と同じ入力があれば，そのうちで最も先頭の入力番号を，なければ 0 を返すプログラム P を設計する．このとき，表 1のテキスト impT のように，まず過去の入力系列を記憶する History 等の補助関数を用いて現在の入力 n に対し，出力 Out を得る状態遷移関数 T を定義する．この状態遷移関数 T の入出力関係のみを射影により取り出し，補助関数を仕様から隠すことができる(表 2テキスト specP の射影文)．初期状態から始めて状態遷移 T を入力系列に対して繰り返すというプログラム P を specP は指定している．

しかし，この方法では要求仕様を，「過去の入力系列中に現在の入力と同じ入力があれば，そのうちのどれでもよいから任意の一つの入力番号を返すプログラム P'」と変えた場合，プログラム P' の関数値は一意に定まらず，射影ではその要求を表せない．そこで，

† 射影文による参照関係がサイクルにならない場合にも，もちろん合同関係は定義できる．ここでは簡単のため本定義を採用する．

表1 状態遷移 T の仕様記述例 1

```
text impT;(* 処理T,Init の実現 *)
(*初期化(イ)の記述 *)
  Out(Init)==0;
  History(Init)==NILL;
  point(Init)==NILL;
(*処理Tのループ関数による展開(ハ)*)
  T(s,n) == UL(UI(s,n),n);
  UL(s,n) == if point(s) = | History(s) | +1 then
UE(s,n)
        else if get(point(s), History(s))= n
        then UF(s,n) else UL(UN(s,n),n);
(*処理 UI,UE,UF,UN の定義(ロ)*)
  Out(UI(s,n))==Out(s);
  History(UI(s,n))==History(s);
  point(UI(s,n))==1;
  Out(UE(s,n))==0;
  History(UE(s,n))==add(History(s),n);
  point(UE(s,n))==point(s);
  Out(UF(s,n))==point(s);
  History(UF(s,n))==add(History(s),n);
  point(UF(s,n))==point(s);
  Out(UN(s,n))==Out(s);
  History(UN(s,n))==History(s);
  point(UN(s,n))==point(s)+1;
end impT;
```

表2 射影によるプログラム P の仕様記述例

```
text specP;
  intC N,O; stateC S;
  Out(T(S,N))==O @text impT; (*処理 Tの仕様*)
  Out(Init)==0;        (*初期化(イ)*)
  include text main; (* main の記述の取り込み*)
end specP;
text main;
  project primitives; (* 基底代数 *)
  Out(L(Init,int_Seq)); (*計算指定項(ニ)*)
(*ループ関数(ハ)*)
  L(s,int_Seq)== if int_Seq = NILL then s
        else L(T(s,car(int_Seq)),cdr(int_Seq));
end main;
```

それが表現できるように拡張射影という概念を導入した[8],[12].

3. 拡張射影を用いたテキストの詳細化の正しさの証明法

3.1 拡張射影の定義

従来のテキストは合同関係を一つ定めていた．このように一つの合同関係で意味定義する方法は他の言語でも普通に行われている．今後，拡張射影を含むテキストを考えるが，これは合同関係の集合を表す．その集合のうちの任意の一つの合同集合を(従来の意味で)満たすように実現すれば良いという意図である．

以下では，テキスト x の表す表現式集合を E_x と表すことにする．

[定義 5] [拡張射影文を含むテキスト u の定める合同関係集合[12]]

テキスト u 中の $\xi_1 == v_1$@@**text** s_1; $\xi_2 == v_2$@@**text** s_2; $\cdots$; $\xi_m == v_m$@@**text** s_n; なる文を拡張射影文と言う．ここで，各 v_j は変数で，ξ_j 中の変数や，v_j には構成子表現式だけを代入可能とする．また，テキスト s_k $(1 \leq k \leq n)$ 自体は，拡張射影文を含まないものとし，$E_u \subseteqq E_{s_k}$ が成り立つとする．上記の拡張射影文において同じ s_k を参照している項 ξ_j は一般に複数存在する．s_k を参照している項の集合を τ_k とする．テキスト t_k をテキスト s_k の項集合 τ_k に関する正しい実現であるとする (t_k は一つの合同関係 $\equiv_{t_k}$ を定める．テキスト s_k と，τ_k を与えたときに，そのようなテキスト t_k は一般に複数存在する)．各 s_k と τ_k について，そのようなテキスト t_k を一つ定めたとき，拡張射影の各 $\xi_j == v_j$@@**text** s_k $(\xi_j \in \tau_k)$ を $\xi_j == v_j$@**text** t_k と読み換えて普通の射影とみなせば，テキスト u の合同関係が一つ定まる．この合同関係を $\equiv_{u[t_1,t_2,\ldots,t_n]}$ と表す．

テキスト u の表す合同関係集合 $\mathcal{R}_u$ は以下のとおり．

$$\mathcal{R}_u = \{\equiv_{u[t_1,t_2,\ldots,t_n]} | \text{各} t_k \text{は} s_k \text{の項集合} \tau_k \text{に関する正しい実現}\}$$

テキスト u が拡張射影を含まなければ $\mathcal{R}_u = \{\equiv_u\}$ とする．

3.2 プログラム P' の拡張射影を用いた仕様例

前章のプログラム P' の仕様は拡張射影を用いて，以下のように記述できる．

テキスト specST に状態遷移関数 T の要求性質を入力変数と番号の対を要素とする集合を表す補助状態成分関数 Set を用いて例えば，「状態遷移関数 T 後の Out の値が 0 と等しくなければ，n と Out の値の対が Set の要素として存在する」等と書いておく(表 3)．プログラム P' の仕様 specP' は，specP の射影文を拡張射影文

Out(T(S,N))==O @@ **text** specST に変更したテキストである．

3.3 拡張射影を含むテキストにおける正しい詳細化

ここで拡張射影を含むテキスト間の正しい詳細化を定義する．

[定義 6] [正しい詳細化[12]] 二つのテキスト $t_1 = (G_1, AX_1)$, $t_2 = (G_2, AX_2)$ において，$E(G_1) \subseteqq E(G_2)$ とする．テキスト t_1 の表す合同関係集合を $\mathcal{R}_1$ とし，テキスト t_2 の表す合同関係集合

を $\mathcal{R}_2$ とする．次の条件を満たすときテキスト t_2 はテキスト t_1 の正しい詳細化であると言い，$t_1 \ll t_2$ と表す．

合同関係集合 $\mathcal{R}_2$ 中のどの合同関係 $\equiv_2$ についても，合同関係集合 $\mathcal{R}_1$ 中にある合同関係 $\equiv_1$ が存在して，$\equiv_2 \supseteq \equiv_1$ である．

3.4 正しい詳細化の十分条件

ASM型記述スタイルで記述したあるテキスト t とそれをASM型記述スタイルで詳細化したテキスト s について，ともに拡張射影を用いているとき，$t \ll s$ であるための十分条件(命題 1)を述べる．まず，前提を置く．

[前提 1] テキスト t, s はそれぞれ，テキスト w, z のみを拡張射影文で参照しており，各テキスト t, s, w, z は，基底代数に関する共通の射影文以外の射影を含まない．この基底代数に関する射影文の記述をPrimitivesとする．これは基本関数の(無矛盾な)定義表を表す．テキスト t (または s) における各状態遷移関数に関する記述は，議論の簡単のため以下の三つの記述のタイプのいずれかであるとする†．

(1) t (または s) に現れるすべての状態成分関数について，「定義」されている．

(2) t (または s) に現れるすべての状態成分関数について，Primitives中の基本関数を用いて「性質記述」されている．

(3) t (または s) に現れるすべての状態成分関数について，以下の拡張射影文が記述されている．その状態遷移を T とする．すべての状態成分関数 F について，

$F(T(s,n)) == v@@$ **text** w (または s 上においては z)

テキスト t から，Primitives とタイプ(3)の記述を除いた残りをASM Common とする．テキスト s は，テキスト t の タイプ(3)の記述の対象となった状態遷移関数のみを展開しているとする．また Primitives と ASM Common を含むものとする．

テキスト t, s の拡張射影文が参照しているテキスト w, z は以下の条件を満たす．テキスト w (または z) は，テキスト t (または s) のタイプ(3)の記述以外のすべての記述および，次の記述からなる．テキスト t (または s) のタイプ(3)の記述の対象となった状態遷移関数に関するタイプ (1),(2) の記述．必要ならば補助的に用いる状態成分関数に関する記述を含んでよい．

テキスト i は以下の公理からなる．w で用いられた各補助状態成分関数に対して，s の(複数の)状態成分関数でどのように対応づけるかを表す公理．これらは補助状態成分関数を s 上でどのように実現するかを表す(ASL)テキストである．対応の記述に用いる関数はすべて，Primitives で定義されているものとする．

Primitives と ASM Common を合わせてテキスト common とする(以上 図 1)．

テキスト t(または s)のタイプ(3)の記述の対象となった項の集合を τ_t(または，τ_s)とする．なお，これらの条件のもとでは，表現式集合について，$E_t \subseteq E_s$ が成り立つ．テキスト i は，必ずしも(そのとおり)実現しなくてもよい補助状態成分関数が，下位のテキスト s でどのように実現されているかを表す公理群で，これは「正しい詳細化であること」の証明時に検証者が与える．

[定義 7] [定理] テキスト t 上の項の対 $\langle \xi, \eta \rangle$ について t 上の許される任意の代入 ρ によって得られる表現式 $\rho(\xi), \rho(\eta)$ に対し，$\rho(\xi) \in E_s$, $\rho(\eta) \in E_s$ が成り立ち，かつ $\rho(\xi) \equiv_s \rho(\eta)$ であるとき，t 上の $\langle \xi, \eta \rangle$ は s 上で定理として成り立つと言い，$\xi \approx_s^t \eta$ と表現する．

以下の命題の証明のために「定理」について以下の補題[9] を用いる．

[補題 1] テキスト t と s はともに拡張射影を含まず，同じ基底代数に対する射影のみをともに含んでいるものとする．また，$E_t \subseteq E_s$ を満たすものとする．

テキスト t 上のすべての公理 $\xi == \eta$ について，$\xi \approx_s^t \eta$ であれば，$\equiv_t \subseteq \equiv_s$ である． □

[命題 1] 前提1の下で，テキスト $(z+i)$ 上でテキスト u の任意の公理が定理として成り立ち，かつ「z の τ_s に関する任意の正しい実現上で，τ_t の各項に t 上で許される任意の代入を行って得られた項が値をもつ」ならば，$t \ll s$ である．

証明: テキスト t, s の表す合同関係の集合をそれぞれ，$\mathcal{R}_t, \mathcal{R}_s$ とする．任意の $\equiv' \in \mathcal{R}_s$ に対して，$\equiv' \in \mathcal{R}_t$ を証明すれば十分である．まず，前提 1 よりテキスト z と w はともに拡張射影および射影を含まず，また同じ基底代数に対する射影のみをともに含んでいる．また，w, z, i の構成より，$E_w \subseteq E_{(z+i)}$ が成り立つ．また，テキスト w の任意の公理は，u にあるか，common にあるかいずれかである．u にあるときは，

† 一般には，s, t において，一つの状態遷移の記述に関して，「定義」と「性質記述」のスタイルが混在する場合についても，正しい詳細化であるための条件を求めることは可能であるが，これらの一般化は本論文では省略する．

表3　状態遷移 T の仕様記述例 2

```
text specST;
  int n; state s;
  AX1:Out(T(s,n))≠0 ⊃ ⟨n,Out(T(s,n))⟩ ∈ Set(s) == TRUE;
  AX2:Out(T(s,n)) = 0 ⊃ ⟨n,i⟩ ∉ Set(s) == TRUE;
  AX3:Set(T(s,n)) = Set(s) ∪{⟨n, |Set(s)| +1⟩} == TRUE;
  AX4:Set(Init) == ∅;
include text main;
end specST;
```

命題1中の条件「テキスト $(z+i)$ 上でテキスト u の公理が定理として成り立つこと」を用い，common にあるときは common の公理がテキスト $z+i$ にもあることを用いて，$E_w \subseteqq E_{(z+i)}$ と補題1より

「$\equiv_w \subseteqq \equiv_{(z+i)}$」$\cdots$ $(*)$

が導ける．また，定義5より，各 $\equiv'$ は，「E_s 中の任意の表現式 ξ,η について $\xi \equiv_z \eta$ ならば $\xi \equiv' \eta$」を満たす．

更に，i は，u 中の補助状態成分関数を独立に s 中の(従って，z 中の)状態成分関数に対応させるだけなので，E_s 中の項については，$\equiv_z$ と $\equiv_{z+i}$ とでは合同関係は変わらない．従って，

各 $\equiv'$ は，「E_s 中の任意の表現式 ξ,η について $\xi \equiv_{(z+i)} \eta$ ならば $\xi \equiv' \eta$」を満たす．$\cdots$ $(**)$.

$E_t \subseteqq E_s$, $(*),(**)$ と，合同関係の満たす関係の推移律より，各 $\equiv'$ は，「E_t 中の任意の表現式 ξ,η について $\xi \equiv_w \eta$ ならば $\xi \equiv' \eta$ を満たす」．一方，命題1の「z の τ_s に関する任意の正しい実現上で，τ_t の各項に t 上で許される任意の代入を行って得られた項が値をもつ」と τ_t の各項に t 上で許される代入をして得られた任意の表現式が E_s に属することより，τ_t 中の項が $\equiv'$ 上で値をもつ．故に，$\equiv' \in \mathcal{R}_t$ である．　□

次に，詳細化の正しさの証明法を上述の十分条件に基づいて与える．

3.5　拡張射影を含むテキストの正しい詳細化の証明法

$E_t \subseteqq E_s$ であることは，順序機械型の詳細化をしていれば成り立つので，$\equiv_t \subseteqq \equiv_s$ であることを示すには，t 上のすべての公理 $\xi == \eta$ について，$\xi \approx_s^t \eta$ であることを示せばよい．

命題1より直ちに次の方法が得られる．

[方法 1]　前提1の下で，(テキスト i を考案し) $t \ll s$ であることを保証するために，u の各公理 $\xi == \eta$ について，$\xi \approx_{(z+i)}^u \eta$ であることを確かめる．「z の τ_s に関する任意の正しい実現上で，τ_t の各項に t 上で許される任意の代入を行って得られた項が値をもつ」ことは，最終的な実現プログラムを得てこの上で無矛盾性，停止性を確かめることによって確認する．

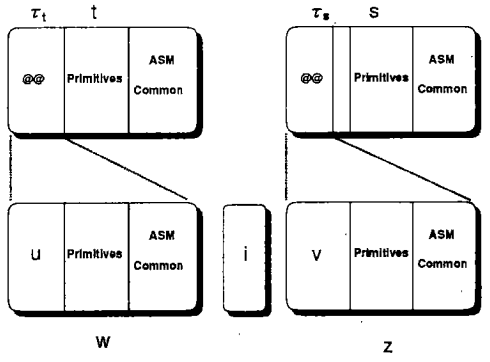


図1　テキストの関係
Fig. 1　Text t and s.

3.6　証 明 例

この例では下位の記述は拡張射影を含まないのでより簡単になる．上位で集合 Set を用いて記述し，下位ではリスト History を用いて実現しているので，これらの関係を与えるテキスト i の公理として例えば，

Set (s) == LS(History(s))

のように，リスト History から 集合 Set への変換関数 LS を用いて表す．LS に関する基本補題として，

LS(add(l,n))= LS(l) $\cup\{\langle n, |\, l\, | +1\rangle\}$ == TRUE

LS(NILL) = $\emptyset$ == TRUE

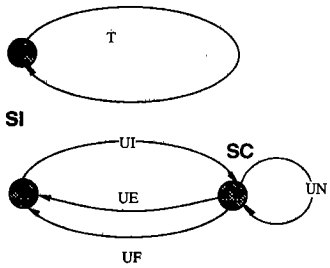
$\langle n,i\rangle \in$ LS(l) $\Leftrightarrow n =$ get(i,LS(l))== TRUE

等が挙げられる．

テキスト (impT+main+i) の合同関係 $\equiv_{(\mathrm{impT+main}+i)}$ と，SpecST の合同関係 $\equiv_{\mathrm{SpecST}}$ について，$\equiv_{(\mathrm{impT+main}+i)} \supseteqq \equiv_{\mathrm{SpecST}}$ であることが証明できる．また，OUT(T(s,n)) が値をもつことは，テキスト (impT+main) 上で T の停止性を証明することにより保証できる．よって，命題1により，specP' $\ll$ (impT+main) である．$\equiv_{(\mathrm{impT+main}+i)} \supseteqq \equiv_{\mathrm{SpecST}}$ であることの証明は，関数の適用に関する帰納法を用いて行う (図2)．

図2は上述の帰納法における証明スキームを表している．状態遷移関数 T の実現が UI, UN, UF, UE を用いて図のような状態遷移で行われている．状態 SC で成り立つであろう不変式 R を検証者が考案し，図にある四つの証明を行う．

例えば公理AX1の場合，C_1, C_2, R, Q としてそれぞれ図2の下段の式を割り当てる．この結果とテキスト i における Set と History の関係式より，公理



Base: $\frac{R(UI(SI),SI,N)}{}$

Ind: $\frac{R(SC,SI,N)\wedge\neg C_1(SC)\wedge\neg C_2(SC,N)}{R(UN(SC,N),SI,N)}$

Final1: $\frac{R(SC,SI,N)\wedge C_1(SC)}{Q(UE(SC,N),SI,N)}$

Final2: $\frac{R(SC,SI,N)\wedge\neg C_1(SC)\wedge C_2(SC,N)}{Q(UF(SC,N),SI,N)}$

For AX1:

$C_1(s) \equiv \mathrm{point}(s) =| \mathrm{History}\ (s) | +1$

$C_2(s,n) \equiv \mathrm{get}(\ \mathrm{point}(s),\ \mathrm{History}\ (s)) = n$

$R(s,t,n) \equiv 1 \leq \mathrm{point}(s) \leq| \mathrm{History}\ (s) | +1$

$Q(s,t,n) \equiv \mathrm{Out}(s) \neq 0 \supset n = \mathrm{get}(\mathrm{Out}(s),\mathrm{History}(t))$

図2 証明図

Fig. 2 Proof of an example program.

AX1が成り立つことが証明できる．

これらの証明を行うための証明支援システムがASLシステムでは用意されている．このシステム上で$\xi \approx \eta$であることを示すためにs上での項書換え，場合分け，数学的決定問題への帰着，補題の追加，構造的帰納法等の方法を用いる[9]．なお，無矛盾性については，プログラムスタイル等より保証できる[9]．

例えば，まず(impT+main+i)のテキストを公理集合として読み込み，次いで各証明スキームの仮定に相当する式を補題追加機能を用いて追加しておく．そして結論部に相当する式ξを既に入力されている公理集合を用いて順次書き換え，必要に応じて場合分けを行ったあと，整数上の恒真性判定機能などを用いて，$\xi \approx^{(\mathrm{SpecST+main})}_{(\mathrm{impT+main+}i)} \mathrm{TRUE}$であることを結論づける．

不変式Rの選択によっては，証明がうまくいかない場合もある．そのときのためにこのような不変式群の変更を容易にできるシステムも作成されている[11]．

4. 到達正当性条件を用いたテキストの詳細化の正しさの証明法

この章では，到達正当性条件を用いたテキストの詳細化の正しさの証明法について述べる．まえがきで述べた理由によりASM型記述の(ホ)のスタイルとして，

$$\mathrm{valid}(s) \wedge \mathrm{Extern}_T(s,l) \wedge \mathrm{select}_T(s,l) \supset p_T(F_k(T(s,l)),F_{k'}(s),l) == \mathrm{TRUE} \quad \text{(V.S)}$$

のように記述する[8]．これは「状態sにおいてプログラムの外部入力lが満たすべき条件$\mathrm{Extern}_T(s,l)$を満足し，かつ，lを引数とする状態遷移関数Tを実行すべき($\mathrm{select}_T(s,l)$が真)ならば，T実行後の状態成分関数値F_k，T実行前の状態成分関数値$F_{k'}$，外部入力lの間には述語p_Tが成り立つ」ことを表している．ここで，Tが外部入力引数をもたなければ，$\mathrm{Extern}(s,l)$の項はない．述語p_Tは実際には，Tの前後の各状態成分関数F_k間の関係を表す述語であるが，以後，「処理Tの前後における関係を表す述語」であることを強調するために，$p_T(T(s,l),s,l)$と状態成分関数を略して記述することもある．一方，各レベルにおいて，「初期状態(Initで表す)でプログラムの外部入力l'が，外部入力が正当であることを表す述語vを満たすときに限り，初期状態のvalidが真になる」よう公理(Ax.1)で定義し，また，「初期状態から始まる実行系列において，各時点の外部入力lが満たすべき条件$\mathrm{Extern}_T(s,l)$を満足しており，かつ，実際に到達した状態に対しては，validが真になる」ように，プログラムの外部入力とプログラムに現われる各処理Tについて公理(Ax.2)で定義する[8]．

$$\mathrm{valid}(\mathrm{Init}(l'))==v(l') \quad \text{(Ax.1)}$$

$$\mathrm{valid}(s)\wedge\mathrm{Extern}_T(s,l)\wedge\mathrm{select}_T(s,l) == \mathrm{valid}(T(s,l)) \quad \text{(Ax.2)}$$

先程と同様にTが外部入力をもたなければ$\mathrm{Extern}_T(s,l)$の項はない．仕様記述者は，$\mathrm{Extern}_T(s,l)$，$v(l')$のみを基本述語等を用いて定義すればよい．select_Tの定義は，以降の節で述べるように，設計者の与えたこのレベルのループ関数によって自動的に定まる．

この記述方法では，外部入力に対する制限Extern，v以外は処理の要求記述時に与えなくてもよい．これにより，まえがきで述べたことが利点となる．

4.1 記述クラス

レジスタ付き有限状態機械とみなせるよう，各ループ関数L_kの記述を以下の形に制限する．

$$\begin{aligned} L_k(s,l) == \ & \text{if } P_1(s,l) \text{ then } C_1 \\ & \text{else if } P_2(s,l) \text{ then } C_2 \\ & \vdots \\ & \text{else } C_m \end{aligned} \quad \text{(L.G)}$$

ここで各C_iは，ループ関数，状態遷移関数からなる状態遷移の系列で，ループ関数L_pの出現は，最外に限られる．また，各$P_i(s,l)$は状態sにおける状態成分関数とlに関する述語である．

この形式にすると，制御部の状態数を有限にできる．以降の議論を簡単にするために，各 C_k を，$L_j(T_i(s,l))$ か，$T_i(s,l)$ の形に制限する((L.G)の形式でループ関数が与えられたとき，ループ関数記号を追加して，この形のループ関数記述に直せるので一般性は失われない)．この制限を以下(L.S)と表す．

4.2 第 k レベルの遷移の選択条件の自動導出

ここでは，第 k レベルの状態遷移関数 T_i を，第 $k+1$ レベルの各状態遷移関数 t_j と，制限(L.S)を満たすループ関数 L_p を用いて展開する際，第 k レベルの T_i の適用前後の制御部の状態名 L_s, L_e と，第 k レベルでの T_i の選択条件 select_{T_i} および，(あれば) T_i の Extern_{T_i} が与えられたとき，(第 $k+1$ レベルの valid の定義の公理で必要となる) 第 $k+1$ レベルで新しく導入された状態名 や，各状態遷移 t_j の選択条件 select_{t_i} をどう与えるかについて述べる†．

但し，状態遷移関数 T_i に外部入力の引数 l がある場合，以下のように，引数 l は，ループ関数で展開される最初の状態遷移関数 t_0 のみが引数とすることにする．すなわち，ループ関数は状態のみを引数とする．

$T_i(s,l) == L_p\ (t_0(s,l))$;

一般に L_p による展開は，状態遷移図の一つの枝を1入口1出口の状態遷移グラフに置き換えることに相当する．第 $k+1$ レベルで新たに導入された状態に対して状態名を表す状態成分関数 name の値を以下のように定める．

t_0 に関する name の公理は

$$\mathrm{valid}(s) \wedge \mathrm{Extern}_{t_0}(s,l) \wedge \mathrm{select}_{t_0}(s,l) \supset \mathrm{name}\ (t_0(s,l)) = \text{if } \mathrm{name}(s)=L_s \text{ then } L_p == \mathrm{TRUE} \quad \ldots\ldots\ldots\ldots\ldots\ldots \text{(N.R0)}$$

となる††．ループ関数 L_p の展開で用いられる各状態遷移 t_i については，

$$\mathrm{valid}(s) \wedge \mathrm{select}_{t_i}(s) \supset \mathrm{name}\ (t_i(s)) = \text{if } \mathrm{name}(s) = L_p \text{ then } L == \mathrm{TRUE} \quad \ldots\ldots\ldots\ldots\ldots\ldots \text{(N.R)}$$

とする．ここで L は以下のように与える．各状態遷移 t_i に対して，この展開において，t_i の出現は，$L_j(t_i(s))$ の形か，$t_i(s)$ のように単独であるかいずれかである．前者の場合 L は L_j とする．後者の場合は L は T_i の適用後の状態名である L_e とする．

t_0 に関する valid の公理で使われる select_{t_0}, Extern_{t_0} の定義は，T_i の select_{T_i}, Extern_{T_i} と同じものにする(S.E)，(図 3)．

一方，t_0 以外の一般の状態遷移関数 t のための select_t は以下のように定義する．

$\mathrm{select}_t(s) ==$ if $\mathrm{name}(s)=L_i$ then $P_i'(s)$ ..(D.S)

ここで，$P_i'(s)$ は，(L_i なる状態において) ループ関数 L_i が t を選択するときの実行条件である．

第 k レベルや，第 $k+1$ レベルにおいて，(同じ名前の状態遷移)T_i, t_j が異なった複数の状態で適用されることがある．異なった出現の T_i ごとに与えられる状態遷移の部分グラフの形は(ループ関数は同一なため)同型である．状態名については，各 T_i ごとに異なった状態名を与えるために，第 $k+1$ レベルの(N.R)の name を与える公理において，ループ関数名 L_j と(第 n 番目の出現を意味する)n の組を name に与える等の変更をすればよい．またこのとき，T_i につき(N.R0)や(N.R)は複数の公理になる．そのときは，if then else のカスケードを用いて，一つの公理にまとめる．これは(D.S)についても同様である．

展開の例を図 3，表 4に挙げておく．ここでは，T_i を公理(L0)，(L1)，(L2)，(L3)のループ関数で展開しており，その場合に対応する状態機械の制御部は図のとおりである．下位のテキストの状態遷移 $t_0, t_1, \ldots, t_5$ の valid の公理は(v0)，...，(v5)であり，また各 select の定義は，(s0),...，(s5)で表される．また各処理の name に関する定義式は公理(n0),...,(n5) に与えられている．

4.3 valid を含むテキストにおける正しい詳細化

このように記述されたテキスト A, B に対して，定義 6のもとで B が A の正しい詳細化であるためには，A にあって B に存在しない次の各公理が，B 上でも定理として成り立てばよい．(I)テキスト A の状態名 name に関する公理，および select の公理，(II)状態遷移関数 T_i に関する valid の定義の公理，(III)テキスト B で展開される T_i について，テキスト A で性質記述(ホ)として使われる公理．

---

† 便宜上第0レベルの記述を考え，この記述は，初期状態 Init の指定と，一つの状態遷移 T とその性質の記述，および計算指定 $F(T(\mathrm{Init}(..),l),l))$ からなるとする．このレベルでは状態遷移 T の前の状態 Init と，状態遷移 T 後の状態 S_F があるとみなせる．第0レベルの各状態における状態名は以下のように $\mathrm{name}(s)$ に保持できる．

$\mathrm{name}\ (\mathrm{Init}(\ldots)) == \mathrm{Init}$

$\mathrm{valid}(s) \wedge \mathrm{Extern}(s,l) \supset \mathrm{name}\ (T(s,l)) = \text{if } \mathrm{name}\ (s) = \mathrm{Init} \text{ then } S_F == \mathrm{TRUE}$

また，このレベルの valid の定義は以下と思ってよい．

$\mathrm{valid}(\mathrm{Init}(\ldots)) == v(\ldots)$

$\mathrm{valid}(s) \wedge \mathrm{Extern}(s,l) \wedge \mathrm{TRUE} == \mathrm{valid}(T(s,l))$

第1レベルの記述は，T を展開して得られる．このときの第1レベルの name や valid の公理は本文の展開則を用いて得られる．

†† if p then g は，p が真と合同であるときのみ，g と合同である．

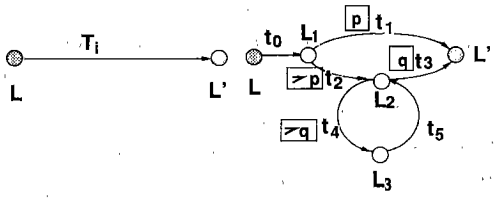


図3 展開の例
Fig. 3 Example of refinement.

表4 validの公理

(vi):valid(s)∧select$_{T_i}$(s,n)∧ Extern$_{T_i}$(s,n)==valid(T_i(s,n))
(v0):valid(s)∧select$_{t_0}$(s,n)∧ Extern$_{t_0}$(s,n)==valid(t_0(s,n))
(v1):valid(s) ∧ select$_{t_1}$(s) == valid(t_1(s))
(v2):valid(s) ∧ select$_{t_2}$(s) == valid(t_2(s))
(v3):valid(s) ∧ select$_{t_3}$(s) == valid(t_3(s))
(v4):valid(s) ∧ select$_{t_4}$(s) == valid(t_4(s))
(v5):valid(s) ∧ select$_{t_5}$(s) == valid(t_5(s))
(L0): T_i(s,n) == L_1(t_0(s,n));
(L1): L_1(s) == if p(s) then t_1(s) else L_2(t_2(s));
(L2): L_2(s) == if q(s) then t_3(s) else L_3(t_4(s));
(L3): L_3(s) == L_2(t_5(s));
(e0):Extern$_{t_0}$(s,n) == Extern$_{T_i}$(s,n);
(s0):select$_{t_0}$(s,n) == select$_{T_i}$(s,n);
(s1):select$_{t_1}$(s) == if name(s)=L_1 then $p(s)$;
(s2):select$_{t_2}$(s) == if name(s)=L_1 then $\neg p(s)$;
(s3):select$_{t_3}$(s) == if name(s)=L_2 then $q(s)$;
(s4):select$_{t_4}$(s) == if name(s)=L_1 then $\neg q(s)$;
(s5):select$_{t_5}$(s) == name(s)=L_3;
(n0):valid(s) ∧ select$_{t_0}$(s,n) ∧ Extern$_{t_0}$(s,n) ⊃
name(t_0(s,n)) = if name(s)=L then L_1 == TRUE;
(n1):valid(s) ∧ select$_{t_1}$(s) ⊃
name(t_1(s)) = if name(s)=L_1 then L' == TRUE;
(n2):valid(s) ∧ select$_{t_2}$(s) ⊃
name(t_2(s)) = if name(s)=L_1 then L_2 == TRUE;
(n3):valid(s) ∧ select$_{t_3}$(s) ⊃
name(t_3(s)) = if name(s)=L_2 then L' == TRUE;
(n4):valid(s) ∧ select$_{t_4}$(s) ⊃
name(t_4(s)) = if name(s)=L_2 then L_3 == TRUE;
(n5):valid(s) ∧ select$_{t_5}$(s) ⊃
name(t_5(s)) = if name(s)=L_3 then L_2 == TRUE;

(II)の公理群は先程の詳細化によって機械的に導出した下位の記述のもとで定理として成り立たない．実際，項の組 ⟨ valid(s)∧ select ⋯ , valid(T(s))⟩ は，下位レベルにおいて T(s) の値が定まらない場合に†，合同関係が満たされない．よって定理として成り立たない．しかし，実用的見地からは，下位レベルにおいては，例えば Extern を満たさない不正な入力に対しては T が停止しないような実現も許されるべきである．そこで，valid を含むテキスト間の詳細化の正しさを次の定義 8 で定義する．この定義では，上位の valid に対する要求を含意(⊃)を用いて，valid(s)∧ select ⋯ ⊃ valid(T(s)) が成り立つこととしている．

[定義 8] [valid を含むテキスト間の詳細化の正しさ] 順序機械型プログラムを**2.1**で述べた詳細化方法で詳細化し，ループ関数の記述クラスは**4.1**の(L.S)を満たし，valid等の定義は**4.2**に従うものとする(記述条件(D.C))．

このようなテキスト A, B に対し，A の valid の定義式(Ax.2)について，

valid(s)∧Extern(s,l)∧select$_T$(s,l) ⊃ valid(T(s,l)) == TRUE ..... (Ax.2')

と読み換えたテキストをテキスト A' とし，テキスト B が A' の正しい詳細化(定義 6)であれば，B が A の正しい詳細化であると言い，この関係を $A \lhd B$ と表す．

関係 $\lhd$ は推移律が成り立つ．この性質は，段階的詳細化では自然に要求される性質である．

4.4 到達正当性条件を含むテキストにおける正しい詳細化の証明法

記述条件(D.C)を満たすテキスト A, B について，$A \lhd B$ を満たすためには，以下が成り立てばよい．

前述の(I)が B 上で定理として成り立つこと．これは，name の定義(N.R)と，記述スタイルの制限(L.S)より自明である．(II)については (Ax. 2')が B 上で成り立つこと．これは命題2で保証する．(III)は後述する．

[命題 2] 記述条件(D.C)を満たすテキスト A, B に対し，テキスト A の性質記述(ホ)で対象となった各状態遷移関数 T_i について valid(s)∧ select$_{T_i}$(s,n)∧ Extern$_{T_i}$(s,n) $\equiv_B$ TRUE を満たす任意の s, n への A 上の許される代入 ρ に対して $\rho(T_i(s,n))$ が B 上で値をもてば††, valid(s)∧ select$_{T_i}$(s,n)∧ Extern$_{T_i}$(s,n) ⊃ valid(T_i(s,n)) $\approx^A_B$ TRUE である．

(証明) 今，議論の簡単のため，テキスト A 上での T_i の出現を1回にしておく．また，テキスト A 上での T_i を行った直後の状態名を L_e と仮定する．

valid(S)∧ select$_{T_i}$(S,N)∧ Extern$_{T_i}$(S,N) $\equiv_B$ TRUE でないことを仮定すると，⊃ の定義より題意が成り立つ．一方B上で TRUE と合同であるとき(⋯(SP))に，valid(T_i(S,N)) $\equiv_B$ TRUE が成り立つことを B 上のループ関数による展開の回数 n に関する帰納法で証明する．

† すなわち，下位レベルのテキスト上の状態遷移関数で，T(s) を展開したときに有限長で表せないとき(以降では状態遷移の値が定まらないというのをこの意味で用いる)．

†† すなわち，「T_i の B 上の遷移による展開が有限長であれば」と同義．

$n=0$ のとき，記述条件(D.C)より valid($T_i(S,N)$) $\equiv_B$ valid($L_k(t_0(S,N))$) であり，このとき，nameの定義(N.R0)より，name($t_0(S,N)$) $\equiv_B$ $L_k(=L_e)$ となる．また，t_0 の select, Extern の定め方(S.E)と仮定(SP)より valid($t_0(S,N)$) $\equiv_B$ TRUE.

さて，一般に $T_i(S,N) \equiv_B L_i(t_n(\cdots t_0(S,N)\cdots))$ であり，valid($t_n(\cdots t_0(S,N)\cdots)$) $\equiv_B$ TRUE かつ，name($t_n(\cdots t_0(S,N)\cdots)$) $\equiv_B$ L_i が成り立つとする(RAS)．今，((L.S)を満たす)ループ関数 L_i を展開する．実行条件式 $P(t_n(\cdots t_0(S,N)\cdots))$ への真偽割当ては有限である．ある真偽割当てで，$T_i(S,N) \equiv_B L_j(t_{n+1}(t_n(\cdots t_0(S,N)\cdots)))$ であれば，selectの定義(D.S)と，帰納法の仮定(RAS)のnameの値より，$\text{select}_{t_{n+1}}$ が真．このとき(RAS)の前半と，Ax.2より，valid($t_{n+1}(t_n(\cdots t_0(S,N)\cdots))$) $\equiv_B$TRUEである．また，name($t_{n+1}(t_n(\cdots t_0(S,N)\cdots))$) の値は，帰納法の仮定(RAS)のnameの与え方と，(N.R)より L_j となる．また，これは，任意の真偽割当てに対し成立する．

valid(s)$\wedge$select$_{T_i}(s,n)\wedge$Extern$_{T_i}(s,n) \equiv_B$ TRUE を満たす任意の s,n への A 上の許される代入 ρ に対して $\rho(T_i(s,n))$ のテキスト B 上での状態遷移の展開が有限であるという仮定より，valid($T_i(s,n)$) $\approx_B^A$ TRUEとなる． □

最後に，(III)について述べる．validの入った式のまま議論できないので，テキスト A 上で状態 s に関して，valid(s) $\supset P(s)$ を満たすと予想される内部状態成分関数間の具体的な言明 $P(s)$ を検証者が考案し，これを用いて証明する方法を用いる．

一般に状態遷移の性質の記述は，本章の最初で述べたように(V.S)の形式で書かれている．この形式において $\supset$ 以降の部分を(処理の)後置条件と呼ぶ．また，$\text{select}_{T_i}(s) \supset \text{select}'_{T_i}(s)$ が成り立ち，かつ容易に検証者が考案できる select'_{T_i} (例えば，実行指定のif文の条件判定式等)を以降で用い，$\text{select}_{T_i}(s) \supset \text{select}'_{T_i}(s)$ が成り立つ性質を適時用いる．

一般にテキスト A(あるいは B)は，(イ)初期化，(ロ)状態遷移の定義の記述，(ハ)ループ関数，(ニ)計算指定および，(ホ)状態遷移の性質の記述からなる．テキスト A'(あるいは，B')を，テキスト A(あるいは B)の(ホ)の各公理をそれぞれ(前提条件をとって)後置条件が真であるという公理に置き換えて A(あるいは B)から得られるテキストとする．

[方法 2] 次の(i)の後，(ii)，(iii) を証明して，テキスト A 上で要求性質が記述された各状態遷移 T_i について，valid(s)$\wedge$ select$_{T_i}(s,l)\wedge$ Extern$_{T_i}(s,l) \supset p_{T_i}(T_i(s,l),s,l) \approx_B^A$ TRUEを結論づける．

(i) 検証者が，テキスト A 上でvalid(s) $\supset P_B(s)$ を満たすと思われる言明 $P_B(s)$ を考案する．

(ii) (i)の言明 $P_B(s)$ に対し，次の(a)，(b)が成り立つこと．

(a) $v(l') \supset P_B(\text{init}(l')) \approx_{A'}^A$TRUE

(b) テキスト A 上の各状態遷移 T_i に対し，$P_B(S)\wedge\text{select}'_{T_i}(S,L)\wedge\text{Extern}_{T_i}(S,L) \equiv_{A'}$TRUE を仮定して，$P_B(T_i(S,L)) \equiv_{A'}$ TRUEであること．

(iii) テキスト A 上の各状態遷移 T_i に対し，テキスト B' 上で，$P_B(S)\wedge\text{select}'_{T_i}(S,L)\wedge\text{Extern}_{T_i}(S,L) \equiv_{B'}$TRUE を仮定して，$p_{T_i}(T_i(S,L),S,L) \equiv_{B'}$TRUEであること．

[命題 3] 方法 2により，テキスト A 上で要求性質が記述された各状態遷移 T_i について，valid(s)$\wedge$ select$_{T_i}(s,l)\wedge$ Extern$_{T_i}(s,l) \supset p_{T_i}(T_i(s,l),s,l) \approx_B^A$ TRUEが成り立つ．

方法 2の利点は次の2点である．テキスト A,B より簡明な A',B' 上で議論できることと，言明 P_B の不変性の証明が，A' 上の大きな状態の単位で行えることである．

命題 3の証明: まず，テキスト A 上の状態を表す項 S の構成に関する帰納法を用いて，

「valid(S) $\equiv_B$ TRUE なる S について，$P_B(S) \equiv_B$TRUEが成り立つこと」…(*)を示す．

基底段階: valid(Init(l')) $\supset P_B(\text{Init}(l')) \equiv_B$ TRUE が成り立つことは，(D.C)および，方法 2 (ii)(a)より，自明．

帰納段階: valid(SC) $\equiv_B$ TRUEなる状態 SC について $P_B(SC) \equiv_B$ TRUEであることを帰納法の仮定とする．

今，valid($T_i(SC,L)$) $\equiv_B$ TRUEを仮定する．このとき，テキスト B 上のvalidの公理(Ax. 2)より valid(SC)$\wedge$ select$_{T_i}(SC,L)\wedge$ Extern$_{T_i}(SC,L) \equiv_B$ TRUEである．よって，このことと，帰納法の仮定，補題 2，および方法 2 (iii)より，

$p_{T_i}(T_i(SC,L),SC,L) \equiv_B$ TRUE (H1)

ここで，状態 SC に関して，T_i の到達正当性条件部が真であることと，(H1)より，A の T_i の処理の性質の公理は，状態 SC においては，B 上で合同関係をもつ．よって，方法 2 (ii)(b)と合わせて，$P_B(T_i(SC,L)) \equiv_B$TRUEが成り立つ．

以上より,「valid($T_i(SC, L)$) $\equiv_B$ TRUEなる状態 $T_i(SC, L)$ について $P_B(T_i(SC, L)) \equiv_B$ TRUE」が成り立つ.

次に,任意のT_iについて,「valid(s)$\wedge$ select$_{T_i}(s, L)\wedge$ Extern$_{T_i}(s, L) \supset p_{T_i}(T_i(s, L), s, L) \approx_B^A$ TRUE が成り立つこと」…(**)を示す.

A 上で定義される任意の状態 SC について,(**)であることは,上述の帰納段階の証明の過程で証明されている.実際,帰納段階の証明において,状態 SC において,valid(SC)$\wedge$ select$_{T_i}(SC, L)\wedge$ Extern$_{T_i}(SC, L) \equiv_B$ TRUEを仮定すると,$p_{T_i}(T_i(SC, L), SC, L) \equiv_B$ が導出される.一方,偽の場合を仮定すると(**)の一つの前半が偽となり,(**)が成り立つ.

すなわち題意が結論される. □

[補題 2] valid(SC) $\wedge$ select$_{T_i}(SC, L)\wedge$ Extern$_{T_i}(SC, L) \equiv_B$ TRUEならば,方法 2(iii)を B' 上で行っても,$p_{T_i}(T_i(SC, L), SC, L) \equiv_B$ TRUEが結論できる.

証明の方針: テキスト B の(Ax. 2)等を用いて証明する. □

4.5 前提条件を陽に記述する方法との比較

提案した手法と従来の(validの代わりに具体的な述語を直接記述する)手法を証明の論法の観点から比較する.従来の手法では,前提条件を(選択条件を含め)具体的に記述している(例えば文献(10)の処理の性質など).従来手法では,T_i をテキストB上で t_i 等を用いて詳細化しているとき,(1)T_i の前提条件 Pre から結論される述語で各 t_i の具体的な前提条件を真にする,状態に関する言明 R を検証者が考案し,(2)言明 R が各 t_i の適用の前の状態で t_i の前提条件を導出すること,(3)言明 R が各 t_i の適用の後の状態で不変式として成り立つこと,(4)各 t_i の要求記述((2),(3)より実際は後置条件のみでよい),R と詳細化のループ関数等を用いてテキスト B 上で T_i の要求が真になること,を証明する.方法 2の(iii)では,(1)における T_i の前提条件 Pre が P_B に対応する.R に対応する不変式も(必要があれば)同様に考案する必要がある.但し,(2)の作業は,本手法では到達正当性条件が補題 2 より自動的に成り立つので不要である.(3),(4)に相当する作業は本手法の(iii)でも必要である.従来の手法では,テキスト A の各状態遷移の展開ごと独立に証明を行う.一方,方法 2では(ii)のような作業が必要である.方法 2の(ii)は,従来の「Pre が確かに T_i の実行前の状態で成り立つことを保証すること」に相当する.従来でも,そのことを保証したければ証明する必要がある.なお,テキスト A が,より上位に対する実現であることを証明する際には,当然 方法 2の(ii)に相当する(2),(3)を行うし,テキスト B の実現を考えるときには,(2),(3)に相当する方法2の(ii)を行うので,行うことはレベルがずれているだけである.また,従来の記述法では,状態遷移の独立性(前提条件を満たすところでは,無条件にその状態遷移を用いられること)があるが,本記述法でも方法 2の(ii)で得られた P_B, Extern$_t$ と select$'_t$ を満たす状態では,t を用いることができ,遜色がない.また本手法で,効率,エラー処理等を考慮して以降の詳細化を行う際にも,この P_B, Extern$_t$ と select$'_t$ を前提条件とみなせばよい.ここで,select$'$ は,前節で定義した select$'$ である.

5. むすび

階層的設計に有用な概念である拡張射影を用いた順序機械型の階層的記述に対する証明方法を提案した.また 到達正当性条件を用いた記述法に対する証明法等を述べた.これらの枠組みにおいて,正しい詳細化を保証する,より広い十分条件の考案等が今後の課題と考えられる.

謝辞 査読者の有益なコメントに感謝致します.

文 献

(1) 稲垣康善,坂部俊樹:"抽象データタイプの代数的仕様記述法の基礎(1)—多ソート代数と等式論理",情報処理,**25**, 1, pp. 47-53 (1984) 他に情報処理,**25**,5, pp. 491-501 (1984) 情報処理,**25**,7, pp. 708-716 (1984) 情報処理,**25**,9, pp. 971-986 (1984).

(2) Wirsing M.: "Handbook of Theoretical Computer Science", Vol.B, Formal Model and Semantics (J. Vanleeuwen Ed.),Chapter 13, Elseveir Science Publishers (1990).

(3) Futatsugi K., Gouguen J., Jounnaud J.-P. and Meseguer J.: "Principles of OBJ2", Symp. of Principles of Programming Languages, pp.52-66 ACM (1985).

(4) Garland S.J. and Guttag J.V.: "An Overview of LP, The Larch Prover", **LNCS 355**, pp.137-151 (1989).

(5) Bergstra J.A., Heering J. and Klint P.: "Algebraic Specification", Acm Press (1989).

(6) 大蘆雅弘,杉山裕二,谷口健一:"代数的言語ASLにおける抽象的順序機械型プログラムとその処理系",信学論(D-I),**J73-D-I**, 12, pp.971-978 (1990-12).

(7) 嵩 忠雄,谷口健一,杉山裕二,関 浩之:"代数的言語ASL/* -意味定義を中心に",信学論(D), **J69-D**, 7,

pp.1066-1075 (1986–07).
(8) 岡野浩三, 北道淳司, 東野輝夫, 谷口健一：“順序機械型プログラムの階層的設計法と在庫管理プログラムの開発例”, 信学論 (D–I), **J76-D-I**, 7, pp.354-363 (1993–07).
(9) 東野輝夫, 関　浩之, 谷口健一：“代数的仕様から関数型プログラムの導出とその実行”, 情報処理, **29**, 8, pp.881-896 (1988).
(10) 岡野浩三, 北道淳司, 東野輝夫, 谷口健一：“代数的言語ASLで記述した在庫管理プログラムとその正しさの証明”, 信学技報, **SS90**-29 (1990).
(11) 森岡澄夫, 北道淳司, 東野輝夫, 谷口健一：“代数的手法を用いた順序機械型プログラムの設計検証”, 信学技報, **SS92**-11 (1992).
(12) 谷口健一：“代数的言語の意味論と代数的手法によるソフトウェア設計開発”, 情報処理研究報告, IF28-1 (1992).

(平成6年4月18日受付，12月26日再受付)

岡野　浩三

平2阪大・基礎工・情報卒．平5同大大学院博士後期課程中退．同年同大情報工学科助手，現在に至る．博士（工学）．代数的手法によるプログラム開発，分散システムなどの研究に従事．情報処理学会会員．

東野　輝夫

昭54阪大・基礎工・情報卒．昭59同大大学院博士課程了．同年同大助手．平2，6モントリオール大学客員研究員．平3阪大・基礎工・情報工学科助教授．工博．分散システム，通信プロトコル等の研究に従事．情報処理学会，IEEE_CS，ACM各会員．

谷口　健一

昭40阪大・工・電子卒．昭45同大大学院博士課程了．同年同大・基礎工・助手，現在，同情報工学科教授．工博．この間，計算理論，ソフトウェアやハードウェアの仕様記述・実現・検証の代数的手法および支援システム，関数型言語の処理系，分散システムや通信プロトコルの設計・検証法などに関する研究に従事．