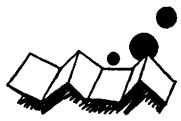
解　説



階層的仕様言語 HISP に基づく変換法プログラミング†

岡　田　康　治††

1. はじめに

プログラミングをより系統的，工学的なものにするため，Wirth[18]，Dijkstra[5]らの構造的プログラミング，Floyd[7]，Hoare[14]の検証法，Dijkstra[6]の最弱事前条件からの構成法など，多くの提案がなされてきた．これら一連の研究には，明らかに，プログラミングを形式的な対象として扱おうとする意図が認められるが，依然として，与えられた問題を完全に仕様記述してからプログラム化を行うという観点には立っていないと考えられる．本稿で紹介する筆者らのプログラミング法は，プログラミングの前半で問題を完全に仕様記述し，後半でその仕様をもとに変換によってプログラムを導出するという，独自の構成をとるものである．

プログラミング全過程の最初に，問題を形式的に定義することの有効性は疑いがない．これにより，意味記述，プログラムの正当性証明，プログラミングの各段階における記述文書など，従来のプログラミングの課題とされてきた諸点が解決される可能性がある．しかし，それはどのようなプログラム作成法として実現されるのか．いかなる仕様言語で，いかに記述すればよいのか．その後に行うプログラム導出とは，今日行われているプログラミングの作業とどう異なるものなのか．何一つとして明らかなものはない．著者らは，それらの答えを求める形で，現在，ある新しいプログラム作成法——**階層的仕様言語に基づく変換法プログラミング**——を開発中である[15),16)]．

次章に詳しく述べるように，このプログラミング法には，陽な形での設計の段階は存在しない．そこで，本稿では，設計の仕事の内容が分かるようにこの方法の概要を紹介することとし，特に，共通例題を仕様記述する過程——ここに，設計の重要な要素である，問題の内容を構造をもった記述に移しかえる作業が含まれる——に重点を置いた．

2 章では，旧来のプログラミング法と比較する形で，階層的仕様言語に基づく変換法プログラミングが提案される．3 章では，同法の前段階である，問題を形式的に仕様記述する過程が，例題に即して詳しく紹介される．4 章では，同法の後段階を成すプログラムの変換について言及される．

2. 新しいプログラム作成法 ——ライフサイクルの否定——

本章では，伝統的なソフトウェアのライフサイクル観に依らない，新しいプログラム作成法を提案する．この方法は信頼性の高いプログラムを系統的に作り出すためのものである．

2.1 旧来のプログラミング

今日まで認められてきた伝統的なプログラム作成過程は，疑似 PASCAL 言語で表現してみると，例えば**図-1** のように表わせるであろう．

ここで，要求定義と仕様記述をまとめてひとつの仕事とするとか，設計を 2 段階でなく 3 段階でやるとかの相違があっても，それは重要ではない．注意すべきは，全過程が逐次的な数種の作業に分かれ，その各作業の内容も各作業が取り扱う文書もまったく異質のものであるということである．ソフトウェア工学では，永い間，その対象分野をこの各作業で細分し，各分野ごとに技法を追求してきた．プログラマは，各作業ごとに技法や知識を，身につけねばならないが，それらは断片的で脈絡のないものが多かった．

指摘しうるもうひとつの点は，どういうプログラムを作るかが，上記の各作業で少しずつ決定されるということである．このプログラミング法では，specify しながらプログラムを書いてゆく．**仕様**という語は，一般に，よく用いられているが，入出力間の写像を定めるものとしてのプログラムを fully specify するも

† Transformational Programming Based upon a Hierarchical Specification Language by Koji OKADA (Computer Science Division, Electrotechnical Laboratory).

†† 電子技術総合研究所ソフトウェア部

```
procedure プログラミング (in x: 問題記述文; out y:
                          プログラム);
      var a: 要求定義書; b: 仕様記述書;
          c: 概要設計書; d: 詳細設計書;
  procedure 要求定義 (in x: 問題記述文; out y:
                      要求定義書); (略);
  procedure 仕様記述 (in x: 要求定義書; out y:
                      仕様記述書); (略);
  procedure 概要設計 (in x: 仕様記述書; out y:
                      概要設計書); (略);
  procedure 詳細設計 (in x: 概要設計書; out y:
                      詳細設計書); (略);
  procedure コーディング (in x: 詳細設計書; out y:
                          プログラム); (略);
  procedure デバッギング (in x: プログラム; out y:
                          プログラム); (略);
begin
      要求定義 (x,a);
      仕様記述 (a,b);
      概要設計 (b,c);
      詳細設計 (c,d);
      コーディング (d,y);
      デバッギング (y,y);
end {プログラミング};
```

図-1 旧来のプログラミング

のを意味していない．したがって，もし，われわれが，プログラムがどうあるべきかを厳密に specify した文書——**真の仕様**——がプログラミングの前にあって，できたプログラムがそれの条件を満たすか否かによってのみプログラムの正当性を論じるという立場をとるならば，今日のプログラミング法で作られるプログラムには正当性の概念は存在しないことになる．真の仕様が存在しない，または，書こうとしないことが，今日のソフトウェア技術の混迷の主要な原因だと思われる．

2.2 変換法プログラミング

本稿で紹介されるプログラム作成法の過程は旧来の方法とはまったく異なっている．疑似 PASCAL では図-2 のように表現される．

ここで前段階の形式的仕様記述とは，形式的な仕様記述言語で問題を完全に記述するもので，これにより，作るべきプログラムは，入，出力の関係を規定するひとつの数学的写像として完全に定義される．後段階の while 文の部分は，**変換**と呼ぶ，プログラムテキストの等価変形の繰返しによって目的のプログラムを導出するものである．

この繰返し型のプログラミングを，**変換法プログラミング(transformational approach for programming)**と呼ぶ（必ずしも上記の議論と一致するわけではないが，類似の方法がいくつか存在する[1)~3)]）．変

```
procedure プログラミング (in x: 問題記述文; out y:
                          形式的仕様);
  procedure 形式的仕様記述 (in x: 問題記述文;
                            out y: 形式的仕様); (略);
  procedure 変換 (in x: 形式的仕様; out y:
                  形式的仕様); (略);
  function 改善度充分 (x: 形式的仕様): Boolean; (略);
begin
      形式的仕様記述 (x,y);
      while not 改善度充分 (y)
      do 変換 (y,y);
end {プログラミング};
```

図-2 変換法プログラミング

換法において，変換の対象となる文書の言語を，その変換法の**基礎言語**と呼ぶ．われわれが本稿で説明しようとしているのは，後述の**階層的仕様言語 HISP** を基礎言語とする変換法である．

変換操作が対象とする各テキストを，版数で区別する．第0版は問題の仕様を意味し，最終版は最終的に得られたプログラムを意味する．変換法では，“仕様”と“プログラム”の語を，伝統的な用法に反して，ある変換操作の入力と出力について，それぞれ，呼ぶことがある．したがって，ある時点のプログラムは，次の変換の仕様となるので，その関係は相対的なものとなり，各版に対して，“より仕様的”，“よりプログラム的”という概念も生じる．

2.3 変換法の条件と利点

変換法によってプログラミングを行うには，その基礎言語が次の条件を満たさねばならない．

1. 言語の構文，意味が厳密に定義されていること．

等価変形を論じるためには，言語の意味が形式的な方法によって定められていなければならない．その前提として，構文も形式的に定められていなければならない．

2. 仕様的記述にも，プログラム的記述にも適していること．

対象問題がいかに大きくとも記述できるように，その構文は，記述の構造化に向いたものであることが求められる．仕様的な記述のためには，本質的に関数の定義で記述を行うことが求められ，プログラム的な記述のためには，構造化された大きな手続きについての表現能力がもとめられる．

3. 実行系を持つ——入力に対して実際に出力を返しうる——こと．

仕様的記述の段階においても実行可能ならば，その記述が記述者の意図を反映しているか否かの検定に利用でき，プロトタイピングの一つの方法として，非常

に有用なので，これも条件に数えることができる．

以上の条件を満たす言語が存在すれば，変換法を用いることができる．変換法には次のような利点が考えられる．

1. 形式的仕様による第0版を基礎として，プログラムの“正当性”や検証の問題を論じうる．
2. 変換は形式的言語の文書上になされるので，自動化しうる部分が非常に大きい．
3. プログラミングの大きな部分が変換の繰返しで一様なので，技法などの開発が容易である．

2.4 階層的仕様言語 HISP の利用

前項の条件を満たす基礎言語として階層的仕様言語 HISP を用いる変換法プログラミングを，筆者らは開発中である．

HISP 言語は次のような性質をもつ．記述の単位はオブジェクトと呼ばれ，記述の全体はオブジェクトのひとつの系列からなる．オブジェクトは，いくつかの関数の定義を束ねたもので，記述すべき概念の適当なまとまりを表わす．オブジェクトをいくつか引用してより高位のオブジェクトを定義することにより，記述全体はモジュール性と階層構造性をもつ．HISP は前項の3条件を満たしていて，この言語を用いる変換法は有効だと考えられる．

この変換法の研究は，形式的仕様記述言語／システム HISP の開発の研究[8)~11)]の発展であり，階層的仕様言語とは，HISP 言語の性質を普通名詞的に呼んだ称である．HISP 言語は，**代数的仕様記述法**[4),12),13),17)]を基礎とし，問題を記述するのに適した構文を整えたものである．現在，HISP システムは，関数を定義する等式を**項書換え規則**と見なす，ある種の実行システムを持っている．その意味で，HISP システムによる関数的プログラミングも可能である．このことが，筆者らが，HISP を基礎言語とする変換法を構想するに至ったひとつの動機となっている．

3. 前段階：形式的仕様記述 ——問題全体の定義——

本章では，変換法の前段階として，共通例題の在庫管理問題を HISP 言語で仕様記述する過程を示す．

3.1 本稿での HISP 言語

HISP は，本来は，英文字で表現される言語であるが，本特集の共通例題の趣旨を生かすため，本稿では仮に，識別子に和文字を許すものとしている．HISP では，オブジェクト名，型名，演算子名の3種の識別子は，それぞれ，BOOL, Bool, true のように最初の2文字の大小によって区別するが，和文字版では，リスト，リスト，長さ，のように区別する．

また，HISP では，演算子名が重複する場合，所属オブジェクト名で“限定”(qualification) して区別することになっているが，本稿では，ある種の規則を仮定して，これを省略できるものとしている．

```
要素:: create sub BOOL
         sort 要素
         op  _=要素 _: 要素, 要素 -> Bool
       end

集合::
create
  sub 要素, BOOL, INT
  sort 集合
  op  空集合: -> 集合
      _: 要素 -> 集合
      _ # _: 集合, 要素 -> 集合
      要素数: 集合 -> Int
      _=集合 _: 集合, 集合 -> Bool
      ...
  eq  var e: 要素; s: 集合
      ( 要素数 (空集合) = 0 )
      ( 要素数 (e) = 1 )
      ( 要素数 (s#e) = 要素数 (s)+1 )
      ...
end

リスト::
create
  sub 要素, BOOL, INT
  sort リスト
  op  空リスト: -> リスト
      _: 要素 -> リスト
      _!_: リスト, 要素 -> リスト
      長さ: リスト -> Int
      _=リスト _: リスト, リスト -> Bool
      ...
  eq  var e: 要素; L: リスト
      ( 長さ (空リスト) = 0 )
      ( 長さ (e) = 1 )
      ( 長さ (L#e) = 長さ (L)+1 )
      ...
end
```

図-3 集合とリスト

また，HISP では，演算子名が重複する場合，所属オブジェクト名で“限定”(qualification) して区別することになっているが，本稿では，ある種の規則を仮定して，これを省略できるものとしている．

3.2 基本的なデータ構造

本問題では，集合およびリストに類似の構造をもったデータ型がよく現われるので，最初に図-3 のように定義した．これは再帰的なデータ構造の HISP における定義法をよく示している．HISP システムは組込みオブジェクトとして BOOL, INT および, STRING

```
品名:: STRING (% String <- 品名 %)

数量:: INT (% Int <- 数量 %)

品ー数::
create
  sub 品名, 数量
  sort 品ー数
  op  [ — — ]: 品名, 数量 -> 品ー数
end
```

図-4 品ー数

を持っていて，上では前二者を利用している．しかし，利用しない自由はあり，任意のオブジェクトを好みのままに定義するのが基本である．例えば，**リスト**では，リストを，最もよく知られた

nil, cons(e, L)

の形ではなく，

空リスト，e, L ! e

の形で定義している．問題向きに設計した一例である．

3.3 名前替え

図-4は，**品名**，**数量**を，それぞれ，STRING, INTの**名前替え**によって定義している．例えば，スタックを，配列とポインタの対で，または，配列と自然数の対で，実現するというとき，ポインタと呼ぶのは，上位概念スタックにおける役割を重視した呼び方であり，自然数とよぶのは，実体を重視した呼び方である．仕様的な記述には前者が適している．**品名**も**数量**も，STRING や INT と，その要素やそれの上の演算は変わらないのだが，あえて，そういう名を持った新しい概念を定義するのである．

品ー数は，品名と数量が対となって使われることが多いので，対化 (pairing) して新しいオブジェクトとしたものである．品ー数型の項の例として，

[“品名1” 5], [“品名2” 13]

などが挙げられる．

3.4 すげ替え

HISP では，**すげ替え**という構文によってパラメータ化データ型を容易に実現できる．**図-5**では**集合**の下位オブジェクトである**要素**を**品ー数**ですげ替えて，**品ー数**の集合という新しいオブジェクトを定義して，そのオブジェクトを**内蔵品集合0**とよんでいる．オブジェクト**内蔵品集合**は，**内蔵品集合0**に，**集合**にはなくて，**内蔵品集合**には必要な演算を追加したものである．例えば，残量という演算は，内蔵品集合と品名とを引数とし，その品の数量を返すものである．HISP

```
内蔵品集合::
refine 内蔵品集合0::
        集合 (* 要素 <- 品ー数;
                  要素 <- 品ー数 *)
             (% 集合 <- 内蔵品集合 %)
  sub 品名, 数量
  op  残量: 内蔵品集合, 品名 -> 数量
      小出し: 内蔵品集合, 品ー数 -> 内蔵品集合
  eq  var n,n0: 品名; i: 数量; ss: 内蔵品集合
  ( 残量 (空集合,n) = 0 )
  ( 残量 ([n0 i],n) = if n0 =str n then i else 0 fi )
  ( 残量 (ss # [n0 i],n )
    = if n0 =str n then i else 0 fi + 残量 (ss,n)
  ( 小出し (...) = ... )
  ...
end
```

図-5 内蔵品集合

```
コンテナ番号:: ...

コンテナ::
create
  sub コンテナ番号, 内蔵品集合
  sort コンテナ
  op  コン: コンテナ番号, 内蔵品集合 -> コンテナ
end
```

図-6 コンテナ

システムは，例えば，

残量([“品名2” 4] # [“品名3” 7],
“品名3”)

という項に対する答えとして，

7

という項を返す．

3.5 レコード型のデータ型

図-6は，コンテナを，コンテナ番号と内蔵品集合の対として定義している．項の例は，

コン(84015, [n0 i0] # [n1 i1])

である．コンテナ番号が5桁の数であることを記述するのは省略した*．この**コンテナ**や**品ー数**の記述に見られるように，いくつかの引数を持つ演算を定義して，特に等式を示さなければ，手続き的プログラミング言語でいう**レコード型のデータ型**の定義となる．

3.6 大きなデータ型

図-7は，**倉庫**をコンテナの集合として定義している．項の例は，

コン(84015, [“品名1” 5] # [“品名2” 3]) #
コン(83880, [“品名1” 28]) #
コン(83572, [“品名2” 10] # [“品名3” 10])

* 以下，問題に対する意見や，記述上の筆者の決定などを文章に下線を付して示す．

```
倉庫::
refine 倉庫0::
        集合 (* 要素<- コンテナ; 要素<- コンテナ *)
             (% 集合<- 倉庫; 空集合<- 空倉庫 %)
  sub  コンテナ番号, 品一数, 品名, 数量, 内蔵品集合
  op   出庫: 倉庫, コンテナ番号, 品一数 ->倉庫
       出庫エラー: ->倉庫
       倉庫残: 倉庫, 品名 ->数量
       コン残: 倉庫, コンテナ番号, 品名 ->数量
       最適コン番: 倉庫, 品名 ->コンテナ番号
  eq   var cn,cn0: コンテナ番号; ni: 品一数;
            ss: 内蔵品集合; n: 品名; s: 倉庫
( 出庫 (空倉庫,cn,ni) = 出庫エラー )
( 出庫 (コン (cn0,ss),cn,ni) = if ... fi )
( 出庫 (s # コン (cn0,ss),cn, ni )
  = if cn0 =int cn
     then s # コン (cn0,小出し (ss,ni))
     else 出庫 (s,cn,ni) # コン (cn0,ss) fi )
( 倉庫残 (空倉庫,n) = 0 )
( 倉庫残 (コン (cn,ss),n) = ... )
( 倉庫残 (s # コン (cn,ss),n)
  = 倉庫残 (s,n) + コン残 (コン (cn,ss),cn,n) )
( コン残 (空倉庫,cn,n) = 0 )
( コン残 (コン (cn0,ss),cn,n) = if ... fi )
( コン残 (s # コン (cn0,ss),cn,n)
  = if cn0 =int cn then 残量 (ss,n)
                   else コン残 (s,cn,n) fi )
( 最適コン番 (...) = ... )
  ...
end
```

図-7　倉庫

である．これを X として，項

　倉庫残 (X, "品名2")

の答えをもとめると，項

　13

となる．

この倉庫は複雑な構造を持ったデータ型で，下位オブジェクトとして，**コンテナ**，**コンテナ番号**，**内蔵品集合**，**品一数**，**品名**，**数量**をもっている．しかし，すでに見てきたように，HISP のオブジェクト定義構文を用いて順に定義することにより，大きな記述が，自然に，階層的に構成されている．

3.7 問題の記述

本稿で記述が求められている受付係の仕事は，倉庫の状態と出庫依頼票とから，出庫指示書と在庫不足票を発行することである．問題に述べられている積荷票の仕様は受付係の仕事と関係がないので省略した．残っている仕様のうち，まず，**出庫依頼票**を図-8に示す．

後に定義する出庫指示書の要素のひとつである注文番号なるものの意味が明らかでないので，出庫依頼票

```
注文番号:: ...

送り先名:: ...

出庫依頼票::
create
  sub  注文番号, 品一数, 送り先名
  sort 出庫依頼票
  op   出依: 注文番号, 品一数, 送り先名 ->出庫依頼票
end
```

図-8　出庫依頼票

```
搬出可否::
refine 搬出可否0:: BOOL (% Bool<- 搬出可否;
                           true<- 可, false<- 否 %)
  sub  倉庫, コンテナ番号, 内蔵品集合
  op   コン搬?: 倉庫, コンテナ番号 ->搬出可否
  eq   var cn,cn0: コンテナ番号; ss: 内蔵品集合; s: 倉庫
( コン搬? (空倉庫,cn) = 否 )
( コン搬? (コン (cn0,ss),cn) = ... )
( コン搬? (s # コン (cn0,ss),cn) = ... )
end

出庫単位::
create
  sub  コンテナ番号, 品一数, 搬出可否
  sort 出庫単位
  op   { _ _ }: コンテナ番号, 品一数, 搬出可否 ->
                出庫単位
end

出庫リスト::
リスト (* 要素<- 出庫単位; 要素<- 出庫単位 *)
       (% リスト<- 出庫リスト;
          空リスト<- 空一出庫リスト %)

出庫指示書::
create
  sub  注文番号, 送り先名, 出庫リスト
  sort 出庫指示書
  op   出指: 注文番号, 送り先名, 出庫リスト ->
             出庫指示書
end
```

図-9　出庫指示書

に含まれているものと仮定して記述した．出庫依頼票の項の例は，

　出依(364, ["品名3" 24], "送り先名1")

である．毎日数十件の依頼を受けることは記述しなかった．

図-9 に出庫指示書を示す．出庫指示書の項の例は，

```
出指(364, "送り先名1",
        { 83245 ["品名3" 14] 可 } !
        { 84002 ["品名3" 8] 可 } !
        { 84173 ["品名3" 2] 否 })
```

```
出庫指示::
refine 出庫指示書
  sub 倉庫，出庫依頼票，コンテナ番号
  op  出庫指示：倉庫，出庫依頼票 -› 出庫指示書
      コン出：倉庫，出庫依頼票，出庫リスト -›
              出庫指示書
      コン出補：倉庫，出庫依頼票，出庫リスト，
              コンテナ番号 -› 出庫指示書
  eq  var s：倉庫；dp：出庫依頼票；n：品名；i：数量；
      a：送り先名；L：出庫リスト；dn：注文番号；
      cn：コンテナ番号
( 出庫指示 (s,dp) = コン出 (s,dp,空－出庫リスト) )
( コン出 (s,出依 (dn,[n i],a),L)
 =if  倉庫残 (s,n) =int 0  then 出指 (dn,a,L)
  else コン出補 (s,出依 (dn,[n i],a),L,
              最適コン番(s,n)) fi )
( コン出補 (s,出依 (dn,[n i],a),L,cn)
 =if コン残 (s,cn,n) < i
  then コン出 (出庫 (s,cn,[n コン残 (s,cn,n)]),
              出依 (dn,[n (i －コン残 (s,cn,n))],a),
              L ! {cn [n コン残 (s,cn,n)]
                   コン搬？(出庫 (s,cn,[n コン残
                            (s,cn,n)]), cn)})
  else 出指 (dn,a,L ! {cn [n i] コン搬？
                  (出庫 (s,cn,[n i]),cn)})
  fi )
end
```

図-10 出庫指示

である．ひとつの出庫指示においては，品名を繰り返し述べる必要はないと思われるが，問題文の書式に従った．

図-10 は**出庫指示**を示している．演算出庫指示は，倉庫と出庫依頼票を引数とし，出庫指示書を返す演算である．コン出―コンテナ単位の出庫処理―，コン出補という補助関数を用いて定義されている．問題文では，注文数に対して在庫が少ないとき，不足の旨を知らせるのみなのか，知らせて，かつ，あるだけを出庫するのか，述べていない．ここでは，仮に，在庫分をすべて出庫することにしておいた．依頼に対してどのコンテナから出していくかを，最適コン番という演算の意味に託して記述してある．この具体的内容については，特に決定しなかったので省略する．

図-11 に**在庫不足検査**を示す．ここで，問題の文は，“在庫不足リストに記入する”ことを求めているようだが，一回の仕事で記入する内容だけを在庫不足票として，それを受付係の出力の一部として定義してみた．リストを引数にとり，一回分の在庫不足票の分を加えたリストを返す演算を定義することは容易な変更である．

図-12 に受付係の仕事を，出庫指示と在庫不足検査

```
在庫不足票::
create
  sub 送り先名，品ー数
  sort 在庫不足票
  op  在不：送り先名，品ー数 -› 在庫不足票
      空ー在不票：-› 在庫不足票
end

在庫不足検査::
refine 在庫不足票
  sub 倉庫，出庫依頼票，品名，数量
  op  在庫不足検査：倉庫，出庫依頼票 -› 在庫不足票
  eq  var s：倉庫；dn：注文番号；n：品名；i：数量；
      a：送り先名
  ( 在庫不足検査 (s，出依 (dn,[n i],a))
   = if 倉庫残 (s,n) < i
     then 在不 (a,(i － 倉庫残 (s,n)))
     else 空ー在不票 fi )
end
```

図-11 在庫不足検査

```
受付::
create
  sub 出庫指示，在庫不足検査，倉庫，出庫依頼票
  sort 受付出力
  op  <＿＿>：出庫指示書，在庫不足票 -› 受付出力
      受付：倉庫，出庫依頼票 -› 受付出力
  eq var s：倉庫；dp：出庫依頼票
  ( 受付 (s,dp) = < 出庫指示 (s,dp)
                    在庫不足検査 (s,dp) > )
end
```

図-12 受付

の対として表現した．HISP では，演算の出力はひとつに限られるので，2つの答えを対化するために，特に受付出力と呼ぶデータ型を用意した．例えば，割算の商と余りや，スタックで，ポップアップされた要素と残りのスタックなどを対で考えるときによく用いられる技法である．

以上で，形式的仕様記述は終った．本章で示されたHISP テキストの全体は，変換の繰返しによって生成されるプログラム列の最初のもの――第0版――を成す．

4. 後段階：プログラムの導出について

変換法の前段階で書かれた仕様は，実行可能な，形式的な言語で書かれているという意味では，すでにプログラムであり，実行されることを意図して書かれたものではないという意味では，まだプログラムとは言えないものである．その点で，曖昧な性質を持つとの印象を与えるかも知れないが，これは，旧来のプログ

ラミングの過程のどこにも現われない，変換法の仕様を，旧来の仕様やプログラムなどの語で理解しようとすることからくる誤解である．変換法においては，変換の繰返しを，プログラムとして所期の条件を満たすものが得られるまで，任意の回数行えばよい．変換法は，プログラムが満たすべき条件や，変換の内容や回数について，何をも規定しない．

変換の例を一つあげる．前章最後のオブジェクト受付において，受付という演算の等式の右辺は，出庫指示と在庫不足検査という同じ引数をもつ２つの演算の結果の対となっている．HISP システムは，これに対して計算を求められると，当然，この２つを独立に計算して答えを対化して返す．これは，プログラムとしては，効率の悪いものであることは明らかである．この２つの演算はトップレベルで似た構造を持っているので，一緒にして計算すべきである．このとき，出庫指示，在庫不足検査，受付の３演算の定義を捨て，前の受付と数学的写像として等しい新しい受付を定義すれば，これが変換の一例である．

しかし，一つの変換ごとに，種々の検討を加えて，プログラミング上のノウハウを適用するのであれば，変換法はあまり魅力的でない．ある意味の自動化を目指して，変換操作のある部分クラスごとに一種の規則として持っているべきである[2),3)]．筆者らは，この観点から，変換操作を厳密に定義し，変換操作の種々の性質を検討し，変換規則を整理することを試みている[16)]．また，ある例題について，変換規則の適用によるプログラム導出の過程を明らかにしている[15)]．これらについては，さらに検討を加え，別な機会に発表したいと考えている．

5. おわりに

本稿では，例題の記述において，一日の処理能力やデータの桁数の制限などの，プログラムの制限条件の記述が困難で割愛することが多かった．これは，本方法を含むオペレーショナル・アプローチ[19)]——ある種のプログラムで仕様を書く——に共通の弱点といえる問題かも知れない．

よく，本方法に対して，“従来技法に比べて，厳密に仕様記述する分の作業が増えるだけではないか”，また，“変換を規則を用いて自動的に行うといっても，本当にオリジナリティを必要とするプログラム改善を機械がやれるとは思えない”との指摘を受ける．実は，筆者らもこれらの点については同様の見通しを持っていて，事実，現在までのいかなる発表においても，工数の大幅な削減や完全な自動化の可能性をうたったりはしていない．本方法の開発の動機は，完全な仕様記述を行わずして，“正しいプログラム”の議論はありえない，という一点にある．

謝辞　研究の機会を与えられた，棟上昭男部長，石井治前部長に感謝する．本稿は，その多くを，鳥居宏次前研究室長，二木厚吉博士との共同研究の成果によっている．ここに記して両氏に感謝の意を表す．

参考文献

1) Bauer, F. L.: Programming as an Evolutionary Process, Proc. of 2nd Int. Conf. on Software Eng., pp. 223-234 (1976).
2) Broy, M. and Pepper, P.: Program Development as a Formal Activity, IEEE Trans. on S. E., Vol. SE-7, No. 1, pp. 14-22 (1981).
3) Burstall, R. M. and Darlington, J.: A Transformation System for Developing Recursive Programs, JACM, Vol. 24, No. 1, pp. 44-67 (1977).
4) Burstall, R. M. and Goguen, J. A.: Putting Theories Together to Make Specifications, Proc. 5th IJCAI, MIT, Cambridge, Mass. (1977).
5) Dijkstra, E. W.: Notes on Structured Programming, Structured Programming, Academic Press (1972).
6) Dijkstra, E. W.: A Discipline of Programming, Prentice-Hall, Inc. (1976).
7) Floyd, R. W.: Assigning Meanings to Programs, Proc. of Symp. in Applied Math., Vol. 19, pp. 19-32 (1967).
8) Futatsugi, K. and Okada, K.: Specification Writing as Construction of Hierarchically Structured Clusters of Operators, Information Processing 80, pp. 287-292 (1980).
9) 二木，岡田：HISP における階層的構造化法，電子通信学会技術研究報告，AL 81-60 (1981).
10) Futatsugi, K. and Okada, K.: A Hierarchical Structuring Method for Functional Software Systems, 6th Int. Conf. on Software Eng., pp. 393-402 (1982).
11) Futatsugi, K.: Hierarchical Software Development in HISP, Computer Science & Technologies 1982, Japan Annual Reviews in Electronics, Computers & Telecommunications Series, OHM-SHA/North-Holland (1982).
12) Goguen, J. A., Thatcher, J. W. and Wagner, E. G.: An Initial Algebra Approach to the Specification, Correctness, and Implementation

of Data Types, IBM Thomas J. Watson Research Center, RC 6487 (#26817) (1976).
13) Guttag, J. V.: The Specification and Application to Programming of Abstract Data Types, Technical Report CSRG-59, Univ. of Toronto (1975).
14) Hoare, C. A. R.: An Axiomatic Basis for Computer Programming, CACM, Vol. 12, No. 10, pp. 576-580, 583 (1969).
15) 岡田，二木，鳥居：変換法によるプログラミング——階層的仕様記述言語に基づく一方法——，電子通信学会技術研究報告，EC 81-76 (1982).
16) 岡田，二木，鳥居：階層的仕様言語による変換プログラミング——変換の型と正当性——，電子通信学会技術研究報告，AL 82-42 (1982).
17) 杉山，谷口，嵩：基底代数を前提とする代数的仕様，電子通信学会論文誌 (D)，Vol. J 64-D, No. 4, pp. 324-331 (1981).
18) Wirth, N.: Program Development by Stepwise Refinement, CACM, Vol. 14, No. 4, pp. 221-227 (1971).
19) Zave, P.: The Operational Versus the Conventional Approach to Software Development, CACM, Vol. 27, No. 2, pp. 104-118 (1984)，邦訳：ソフトウェア開発の新方式として現れたオペレーショナル・アプローチ，日経エレクトロニクス，No. 345, pp. 235-258 (1984).

(昭和 59 年 10 月 3 日受付)