Vol. 37 No. 4　　情　報　処　理　　Apr. 1996

解　説

集合論プログラミング†

野　木　兼　六††

1. はじめに

計算機の急速な発展にともない，ソフトウェアはますます複雑化・高度化し，その開発はきわめて困難になりつつある．このような“ソフトウェアの危機”を解決するために，1970年代にソフトウェア工学が提唱されたが，その後の多くの研究にもかかわらず，保守問題の究極的な解決という当初の目標はいまだに達成されていない．その最大の理由は，これまでのソフトウェア工学が仕様とプログラムの間に存在する大きなギャップをうまく克服することができなかったためといえよう．プログラミングの難しさの多くは，それが“問題の世界”と“計算の世界”という2つの異質な世界に係わっていることに起因している．“問題の世界”とは「何を」するかを考える世界であり，そこでは宣言的な思考が要求される．一方，“計算の世界”とは「いかに」するかを考える世界であり，ここでは手続き的な思考が要求される．仕様は“問題の世界”に属し，プログラムは“計算の世界”に属するため，仕様からプログラムへの変換はきわめて困難な作業になる．また，これまでのプログラミングでは，仕様は自然言語や図表などを使って非形式的に記述されることが多く，仕様からプログラムへの変換はさらに困難な作業になっていた．曖昧な仕様から正しいプログラムを合理的に導出することはできないのである[1]．

このような状況を改善するために，1980年代には，関数型プログラミング，論理型プログラミング，オブジェクト指向プログラミングなどの新しいプログラミングパラダイムが提案された．これらは，いずれも，プログラムをより“問題の世界”に近い世界（の言葉や概念）で表現し，仕様とプログラムの間に存在するギャップを何とか克服しようとする試みとも考えられる．ただし，これらは，あくまでも，プログラミング言語という枠組の中で考えられているため，仕様記述言語という面から見ると，必ずしも満足すべきものにはなっていない．仕様はできるだけ宣言的であって欲しいが，宣言的になるほど実行効率が悪くなるため，仕様記述言語とプログラミング言語は，一般に，両立しないのである．

以上のような理由から，近年，プログラミング言語から独立した形式的仕様記述言語に対する関心がますます高まりつつある．形式的仕様に関する研究には2つの流れがあった．1つは代数的仕様であり，もう1つは論理的仕様である．代数的仕様は，抽象データ型の形式的仕様を等式を使って記述するものであるが，どちらかというと，低レベルの形式的仕様であり，プログラミング言語に近い．一方，論理的仕様は，関数や手続きの形式的仕様を論理式や集合を使って記述しようとするものであり，より高レベルの形式的仕様といえる．人間にとって分かりやすい形式的仕様を記述するためには，低レベルの代数的仕様だけでは不十分であり，どうしても，高レベルの論理的仕様が必要になる．このため，1990年頃から，代数的仕様と論理的仕様を融合した本格的な形式的仕様記述言語が追求されるようになった[2),3)]．

本稿では，以上で述べたような動向を踏まえつつ，形式的仕様からプログラムを合理的に導出することを目指した新しいプログラミングパラダイムの1つの候補として，集合論プログラミングを概観し，その一般性と有効性を論じる[4)]．

2. 集合論プログラミングの概略

本章では，集合論プログラミングのイメージを簡単な例に沿って説明する．例としては，空でな

† Set Theoretic Programming by Kenroku NOGI (Advanced Research Laboratory, Hitachi, Ltd.).
†† 日立製作所基礎研究所

い整数の有限集合の最大値を求める問題を取り上げる．集合論プログラミングでプログラムを開発するには，まず，集合論仕様を記述する必要がある．集合論仕様は，公理的集合論に基づく形式的仕様であり，プログラムを合理的に開発するための基礎を与える．最大値問題の集合論仕様は次のようになる．

$$\begin{aligned}&\mathrm{Max}(X\in\mathrm{Set(Int)}\mid X\neq\{\})\\&\quad\equiv\{y\in X\mid\mathrm{Largest}(y,X)\},\\&\mathrm{Largest}(y\in U,U\in\mathrm{Set(Int)})\\&\quad=\vee u\in U[y\geq u]\end{aligned}$$

ここで，Max は定義すべき関数名，Largest は述語名を表す．また，Set(Int) は整数の有限集合の全体から成る集合を表す．この例からも分かるように，集合論仕様は，集合記法で問題を素直に表現したものになっており，直観的に理解しやすい．

集合論プログラミングでは，次に，宣言的な集合論仕様から再帰的プログラムを導出する．その基本的な考えは，まず，再帰性のパターンを適当に選択し，次に，このパターンに沿って宣言的な集合論仕様を再帰的な集合論仕様に等価変換し，最後に，再帰的な集合論仕様からこれを満足する再帰的プログラムを抽出するというものである．なお，任意の入力データ x に対し，$\mathrm{p}(x)\in\mathrm{P}(x)$ が成り立つとき，再帰的プログラム p は集合論仕様 P を満足するという．（以下では，再帰的プログラム名として集合論仕様名の小文字版を使う．）最大値問題に対する集合論仕様の等価変換過程は次のようになる．

$$\begin{aligned}&\mathrm{Max}(\{x\})\\&\quad=\{y\in\{x\}\mid\mathrm{Largest}(y,\{x\})\}\\&\quad=\{y\in\{x\}\mid y\geq x\}\\&\quad=\{x\}\\&\mathrm{Max}(X_1\cup X_2)\\&\quad=\{y\in X_1\cup X_2\mid\mathrm{Largest}(y,X_1\cup X_2)\}\\&\quad=\{y:y\in\{y_1,y_2\},y_1\in X_1,y_2\in X_2\mid\\&\qquad\mathrm{Largest}(y,X_1\cup X_2)\}\\&\quad=\{y:y\in\{y_1,y_2\},y_1\in X_1,y_2\in X_2\mid\\&\qquad\mathrm{Largest}(y_1,X_1)\wedge\mathrm{Largest}(y_2,X_2)\wedge\\&\qquad\mathrm{Largest}(y,X_1\cup X_2)\}\\&\quad=\{y:y\in\{y_1,y_2\},y_1\in X_1,y_2\in X_2\mid\\&\qquad y\geq y_1\wedge y\geq y_2\wedge\\&\qquad\mathrm{Largest}(y_1,X_1)\wedge\mathrm{Largest}(y_2,X_2)\}\\&\quad=\{y:y\in\{y_1,y_2\},y_1\in\mathrm{Max}(X_1),\\&\qquad y_2\in\mathrm{Max}(X_2)\mid y\geq y_1\wedge y\geq y_2\}\end{aligned}$$

この再帰的な集合論仕様から次のような再帰的プログラムが抽出される．

$$\begin{aligned}&\max(\{x\})\\&\quad=\mathrm{any}(\mathrm{Max}(\{x\}))\\&\quad=\mathrm{any}(\{x\})\\&\quad=x\\&\max(X_1\cup X_2)\\&\quad=\mathrm{any}(\mathrm{Max}(X_1\cup X_2))\\&\quad=\mathrm{any}(\{y:y\in\{y_1,y_2\},y_1\in\mathrm{Max}(X_1),\\&\qquad y_2\in\mathrm{Max}(X_2)\mid\\&\qquad y\geq y_1\wedge y\geq y_2\})\\&\quad=\mathrm{any}(\{y\in\{y_1,y_2\}\mid y\geq y_1\wedge y\geq y_2\})\\&\qquad\text{where } y_1=\mathrm{any}(\mathrm{Max}(X_1)),\\&\qquad\quad y_2=\mathrm{any}(\mathrm{Max}(X_2))\\&\quad=y_1\quad\text{if } y_1\geq y_2\\&\qquad y_2\quad\text{if } y_2\geq y_1\\&\qquad\text{where } y_1=\max(X_1),\ y_2=\max(X_2)\end{aligned}$$

ここで，any(a) は集合 a の任意の要素を表す．

以上により，最大値問題の再帰的プログラムが得られた．最大値問題は簡単であったため，集合論仕様から直接効率の良い再帰的プログラムが得られたが，効率が悪い場合には，さらに最適化する必要がある．

3. 集合論仕様

本章では，集合論仕様の概略を説明する[5]．集合論仕様は，関数や述語の定義の集まりである．定義は“左辺≡右辺”の形をしており，左辺はそこで定義される関数や述語の名前とその引数，右辺は対応する集合式や論理式である．集合式は，∩ や ∪ などの集合演算を含む常識的な集合式であるが，特に，次のような集合式が許される．

$$\{\mathrm{E}(y,x):y\in\mathrm{F}(x)\mid\mathrm{Q}(y,x)\}$$

ここで，x は自由変数（引数），y は束縛変数，$\mathrm{F}(x)$ は集合式，$\mathrm{Q}(y,x)$ は論理式，$\mathrm{E}(y,x)$ は要素値をとる集合式を表す．この集合式は，$\mathrm{Q}(y,x)$ を満足する集合 $\mathrm{F}(x)$ の要素 y の全体から成る集合の $\mathrm{E}(y,x)$ による像を表示する．$\{y:y\in\mathrm{F}(x)\mid\mathrm{Q}(y,x)\}$ は $\{y\in\mathrm{F}(x)\mid\mathrm{Q}(y,x)\}$，$\{\mathrm{E}(y,x):y\in\mathrm{F}(x)\mid\mathrm{true}\}$ は $\{\mathrm{E}(y,x):y\in\mathrm{F}(x)\}$ と省略してもよい．この集合式は，$\mathrm{F}(x)$ が無限集合の場合にも意味を持つが，特に，$\mathrm{F}(x)$ が有限集合の場

合には，いわゆる，生成&テスト形と見なすことができるので，Fを生成関数，Qをテスト述語と呼ぶことにする．論理式は，∧，∨，¬，⇒，⇔，∀，∃などの論理演算，∈，=，⊂などの述語を含む常識的な（1階述語）論理式である．

集合論仕様の最も大きな特徴は，実世界における種々の概念を自然に定義することができることである．概念を定義する方法には様々なものがあるが，最も代表的な方法は類概念と種差による定義であろう．

概念＝類概念＋種差

すなわち，一般化や特殊化によって生じる概念の系列から2つの隣接した概念を取り出したとき，外延の広いほうを類概念，狭いほうを種概念といい，ある種概念を同一の類概念に属する他の種概念から区別する特徴を種差という．たとえば，“親友とは親密な友人である”という例では，“親友”という概念が“友人”という類概念と“親密な”という種差によって定義されている．集合記法でこれを表現すれば，次のようになる．

親友$(x\in$人$)\equiv\{y\in$友人$(x)|$親密な$(y, x)\}$

すなわち，生成関数Fが類概念に相当し，テスト述語Qが種差に相当する．集合論仕様では，出力データという概念を概念の系列に沿って階層的に定義していけば，自然に全体の仕様が得られる．

集合論仕様の例を以下に示す．なお，本稿で使われる標準的なデータ型と関数を付録に示しておく．

例3.1 商と余り

$$\mathrm{Div}(x\in\mathrm{Nat}, y\in\mathrm{Nat}|y\neq 0)$$
$$\equiv\{\langle q, r\rangle:q\in 0..x, r\in 0..<y|x=q*y+r\}$$

例3.2 順探索

$$\mathrm{Search}(x\in X, r\in\mathrm{Seq}(X)|r\neq[]\wedge x\in\mathrm{rng}(r))$$
$$\equiv\{i\in 0..<|r||\forall j\in 0..<i[x\neq r(j)]\wedge x=r(i)\}$$

例3.3 ソーティング問題

$$\mathrm{Sort}(r\in\mathrm{Seq}(\mathrm{Int}))\equiv\{s\in\mathrm{Perm}(r)|\mathrm{Ord}(s)\},$$
$$\mathrm{Ord}(s\in\mathrm{Seq}(\mathrm{Int}))\equiv\forall i\in 0..<|s|-1[s(i)\leqq s(i+1)],$$
$$\mathrm{Perm}(r\in\mathrm{Seq}(X))\equiv\{s\in\mathrm{Seq}(|r|, \mathrm{rng}(r))|\mathrm{Relative}(s, r)\},$$
$$\mathrm{Relative}(s\in\mathrm{Seq}(X), r\in\mathrm{Seq}(X)||s|=|r|)\equiv\exists h\in\mathrm{Seq}(|s|, \mathrm{dom}(r))[\mathrm{rng}(h)=\mathrm{dom}(r)\wedge\forall i\in 0..<|s|[s(i)=r(h(i))]]$$

例3.4 最長上昇列問題

$$\mathrm{LUpSeq}(r\subset\mathrm{Seq}(\mathrm{Int}))\equiv\{|s|:s\in\mathrm{UpSeq}(r)|\ \mathrm{Longest}(s, \mathrm{UpSeq}(r))\},$$
$$\mathrm{Longest}(s\in U, U\in\mathrm{Set}(\mathrm{Seq}(X)))\equiv\forall u\in U[|s|\geqq|u|],$$
$$\mathrm{UpSeq}(r\in\mathrm{Seq}(\mathrm{Int}))\equiv\{s\in\mathrm{SubSeq}(r)|\mathrm{Up}(s)\},$$
$$\mathrm{Up}(s\in\mathrm{Seq}(\mathrm{Int}))\equiv\forall i\in 0..<|s|-1[s(i)\leqq s(i+1)]$$
$$\mathrm{SubSeq}(r\in\mathrm{Seq}(X))\equiv\cup\{\mathrm{SubSeq}(k, r)|k\in 0..|r|\}$$
$$\mathrm{SubSeq}(k\in 0..|r|, r\in\mathrm{Seq}(X))\equiv\{s\in\mathrm{Seq}(k, \mathrm{rng}(r))|\mathrm{Sub}(s, r)\}$$
$$\mathrm{Sub}(s\in\mathrm{Seq}(X), r\in\mathrm{Seq}(X)||s|\leqq|r|)\equiv\exists h\in\mathrm{Seq}(|s|, \mathrm{dom}(r))[\forall i\in 0..<|h|-1[h(i)<h(i)]\wedge\forall i\in 0..<|s|[s(i)=r(h(i))]]$$

4. 構造帰納法に基づくアルゴリズムの発見

本章では，集合論仕様から再帰的プログラムを導出するための具体的な方法として，構造帰納法に基づくアルゴリズムの発見法を説明する[5]．

4.1 データとプログラムの再帰的構造

仕様からプログラムへの変換過程には様々な局面が含まれているが，その中で最も創造的な過程はアルゴリズムの発見であろう．宣言的な仕様を手続き的なプログラムに変換するためには，どこかで手順的な思考を導入しなければならない．そのきっかけを与えるものがアルゴリズムである．ところで，チャーチのテーゼによれば，アルゴリズムとは再帰的関数のことであるから，アルゴリズムの発見とは再帰性の導入にほかならないともいえる．

アルゴリズム＝仕様＋再帰性

すなわち，（宣言的な）仕様に再帰性を導入すれ

(1) 左端逐次構造
$[x_1[x_2[x_3[x_4[\,]]]]]$
(2) 右端逐次構造
$[[[[[\,]x_1]x_2]x_3]x_4]$
(3) 2分構造
$[[[x_1][x_2]][[x_3][x_4]]]$
(4) 変形2分構造
$[[[\,]x_1[\,]]x_2[[\,]x_3[\,]x_4[\,]]]$
図-1 データ(例)の再帰的構造

ば，アルゴリズムが得られる．もちろん，プログラムは単に計算すればよいというものではないから，より効率の良いアルゴリズムを発見することが重要であり，そのためには，種々の再帰性のパターンを用意する必要がある．

集合論プログラミングでは，次のようなデータ(たとえば，列）の再帰的構造を考える．これを**図-1**に示す．なお，以下の再帰的定義では，データの大きさ（長さ）をnで表す．

（1） 左端逐次構造
$n=0$のとき，$t=[\,]$
$n>0$のとき，$t=[x]\|t'$
（2） 右端逐次構造
$n=0$のとき，$t=[\,]$
$n>0$のとき，$t=t'\|[x]$
（3） 2分構造
$n\leqq 1$のとき，$t=[\,]$，$t=[x]$
$n>1$のとき，$t=t_1\|t_2$（$t_1\neq[\,]$，$t_2\neq[\,]$）
（4） 変形2分構造
$n=0$のとき，$t=[\,]$
$n>0$のとき，$t=t_1\|[x]\|t_2$

集合論プログラミングでは，このようなデータの再帰的構造に対し，次のようなプログラムの再帰的構造を考える．

（1） 左端逐次構造
$n=0$のとき，$P(r)=P_0(r)$
$n>0$のとき，$P(r)=\{s:s\in P_{out}(x,s',r'),\ s'\in P(r'),\langle x,r'\rangle\in P_{in}(r)\}$
（2） 右端逐次構造
$n=0$のとき，$P(r)=P_0(r)$
$n>0$のとき，$P(r)=\{s:s\in P_{out}(s',x,r'),\ s'\in P(r'),\langle r',x\rangle\in P_{in}(r)\}$
（3） 2分構造
$n\leqq 1$のとき，$P(r)=P_0(r)$，$P(r)=P_1(r)$
$n>1$のとき，$P(r)=\{s:s\in P_{out}(s_1,s_2,r_1,r_2),\ s_1\in P(r_1),\ s_2\in P(r_2),\ \langle r_1,r_2\rangle\in P_{in}(r)\}$
（4） 変形2分構造
$n=0$のとき，$P(r)=P_0(r)$
$n>0$のとき，$P(r)=\{s:s\in P_{out}(s_1,x,s_2,r_1,r_2),\ s_1\in P(r_1),\ s_2\in P(r_2),\ \langle r_1,x,r_2\rangle\in P_{in}(r)\}$

これらの再帰的構造は，いずれも，まず，P_{in}によって入力データrをデータの構成に関してより次数の低いデータに分解し，次に，再帰的な計算を行い，最後に，P_{out}によってデータの構成に関してより次数の低いデータから出力データsを合成するものになっている．

以上のようなデータの構成に関する再帰的構造のほかに，データの位相に関する再帰的構造があるが，説明は省略する[5]．

4.2 集合論仕様の等価変換

構造帰納法は，半整列集合上で定義された命題を証明するための手法である[6]．この構造帰納法を使って，半整列集合を扱うプログラムの種々の性質，たとえば，正当性，等価性などを証明することができる．集合論プログラミングでは，集合論仕様に再帰性を導入するための具体的な方法として，この構造帰納法を利用する．

再帰性＝構造帰納法

すなわち，前項で述べたようなデータの再帰的構造によって，列などの基本的なデータ領域は半整列集合になるので，前項で述べたような再帰的構造を持つプログラムが与えられれば，構造帰納法を使って，その正当性を証明することができる．したがって，正しいプログラムを導出するためには，逆に，正当性の証明が容易になるように，再帰性のパターンを決定していけばよい．

集合論仕様$P(r)\equiv\{s\in F(r)|Q(s,r)\}$に再帰性を導入するための手順を以下に示す．ただし，$n=0$の場合は，単純な等価変換になるので，省略し，$n>0$の場合のみを考える．また，簡単のために，2分構造を例にとって説明する．なお，実際の変換には，展開/畳み込み方式のプログラム変換手法を利用する[7]．

【ステップ1】展開
$P(r)$を展開する．
$P(r)$

$=\{s\in F(r)|Q(s,r)\}$

【ステップ2】生成関数の分解

Fの分解形は，次の条件を満足するとき，Qに関して生成的であるという．

$\{s\in F(r)|Q(s,r)\}$
$\subset\{s:s\in F_{out}(s_1,s_2,r_1,r_2),$
$s_1\in F(r_1),\ s_2\in F(r_2),\ \langle r_1,r_2\rangle\in F_{in}(r)\}$
$\subset F(r)$

Fの分解形がFの再帰形になっていれば，任意のQに関して生成的である．このとき，次の等式が成り立つ．

$\{s\in F(r)|Q(s,r)\}$
$=\{s:s\in F_{out}(s_1,s_2,r_1,r_2),$
$s_1\in F(r_1),\ s_2\in F(r_2),\ \langle r_1,r_2\rangle\in F_{in}(r)|$
$Q(s,r)\}$

そこで，まず，Qに関して生成的なFの分解形を決定する．

【ステップ3】許容性による変換

Fの分解形は，次の条件を満足するとき，Qに関して許容的であるという．

$\{s\in F(r)|Q(s,r)\}$
$\subset\{s:s\in F_{out}(s_1,s_2,r_1,r_2),$
$s_1\in F(r_1),\ s_2\in F(r_2),\ \langle r_1,r_2\rangle\in F_{in}(r)|$
$Q(s_1,r_1)\wedge Q(s_2,r_2)\}$

このとき，次の等式が成り立つ

$\{s\in F(r)|Q(s,r)\}$
$=\{s:s\in F_{out}(s_1,s_2,r_1,r_2),$
$s_1\in F(r_1),\ s_2\in F(r_2),\ \langle r_1,r_2\rangle\in F_{in}(r)|$
$Q(s_1,r_1)\wedge Q(s_2,r_2)\wedge Q(s,r)\}$

そこで，Fの分解形がQに関して許容的ならば，テスト部に$Q(s_1,r_1)\wedge Q(s_2,r_2)$を追加する．

【ステップ4】テスト述語の分解

簡略化によってテスト述語Qを次のように分解する．

$\{s\in F(r)|Q(s,r)\}$
$=\{s:s\in F_{out}(s_1,s_2,r_1,r_2),$
$s_1\in F(r_1),s_2\in F(r_2),\langle r_1,r_2\rangle\in F_{in}(r)|$
$Q_{out}(s,s_1,s_2,,r_1,r_2)\wedge$
$Q(s_1,r_1)\wedge Q(s_2,r_2)\wedge Q_{in}(r_1,r_2,r)\}$

【ステップ5】畳み込み

$P(r_1)$と$P(r_2)$を畳み込む．

$P(r)$
$=\{s:s\in P_{out}(s_1,s_2,r_1,r_2),$
$s_1\in P(r_1),\ s_2\in P(r_2),\ \langle r_1,r_2\rangle\in P_{in}(r)\}$
where $P_{out}(s_1,s_2,r_1,r_2)$
$=\{s\in F_{out}(s_1,s_2,r_1,r_2)|$
$Q_{out}(s,s_1,s_2,,r_1,r_2)\}$,
$P_{in}(r)$
$=\{\langle r_1,r_2\rangle\in F_{in}(r)|$
$Q_{in}(r_1,r_2,r)\}$

集合論仕様が階層化されている場合には，以上の手順を階層に沿って再帰的に適用する．ただし，F_{in}とF_{out}，Q_{in}とQ_{out}には任意性があり，どちらかを固定しないと，他方を決定することができない．このことから，以下のような構造帰納法が得られる．

（1） 入力帰納法

入力帰納法は入力データの自然な構成に関する構造帰納法である．すなわち，F_{in}を前項で述べたような入力データの自然な分解，Q_{in}を無条件とし，これからF_{out}とQ_{out}を決定する．

例4.1 順探索（例3.2）

$Search(x,[])$
$=$undefined
$Search(x,[y]\|r)$

【ステップ1】

$=\{i\in 0..<|[y]\|r|\|$
$\forall j\in 0..<i[x\neq[y]\|r(j)]\wedge x=[y]\|r(i)\}$

【ステップ2】

$=\{i:i\in\{0,1+i'\},\ i'\in 0..<|r|\|$
$\forall j\subset 0..<i[x\neq[y]\|r(j)]\wedge x=[y]\|r(i)\}$

【ステップ3】

$=\{0|x=y\}\cup\{1+i:i\in 0..<|r|\|$
$\forall j\in 0..<i[x\neq r(j)]\wedge x=r(i)\wedge$
$\forall j\in 0..<i+1[x\neq[y]\|r(j)]\wedge$
$x=[y]\|r(i+1)\}$

【ステップ4】

$=\{0|x=y\}\cup\{1+i:i\in 0..<|r|\|$
$x\neq y\wedge\forall j\in 0..<i[x\neq r(j)]\wedge x-r(i)\}$

【ステップ5】

$=\{0:|x=y\}\cup\{1+i:i\in Search(x,r)|x\neq y\}$

（2） 出力帰納法

出力帰納法は出力データの自然な構成に関する構造帰納法である．すなわち，F_{out}を前項で述べたような入力データの自然な合成，Q_{out}を無条件とし，これからF_{in}とQ_{in}を決定する．

例4.2 商と余り（例3.1）

$Div(x,y)$

$=\{\langle 0, r\rangle : r\in 0..<y \mid x=r\}$
$-\{\langle 0, x\rangle \mid x<y\}$
Div(x, y)

【ステップ1】
$=\{\langle 1+q, r\rangle : 1+q\in 0..x, r\in 0..<y \mid$
$x=(1+q)*y+r\}$

【ステップ2】
$=\{\langle 1+q, r\rangle : q\in 0..x', x'\in 0..x-1, r\in 0..<y \mid$
$x=y+q*y+r\}$

【ステップ3】
$=\{\langle 1+q, r\rangle : q\in 0..x', x'\in 0..x-1, r\in 0..<y \mid$
$x'=q*y+r\wedge x-y=q*y+r\}$

【ステップ4】
$=\{\langle 1+q, r\rangle : q\in 0..x', x'\in 0..x-1, r\in 0..<y \mid$
$x'=q*y+r\wedge x'=x-y\}$
$=\{\langle 1+q, r\rangle : q\in 0..x-y, r\in 0..<y \mid$
$x-y=q*y+r\wedge x\geqq y\}$

【ステップ5】
$=\{\langle 1+q, r\rangle : \langle q, r\rangle \in \mathrm{Div}(x-y, y) \mid x\geqq y\}$

以上のようなデータの構成に関する構造帰納法のほかに，データの位相に関する構造帰納法（収束帰納法）があるが，説明は省略する[5]．

4.3 再帰的プログラムの抽出

すでに集合論仕様が再帰的な形に変換されているため，これから再帰的プログラムを容易に抽出することができる．

例4.3 順探索

search$(x, [])$
$=$undefined
search$(x, [y]\|r)$
$=$any(Search$(x, [y]\|r)$)
$=$any$(\{0: \mid x=y\}\cup\{1+i: i\in$Search$(x, r) \mid$
$x\neq y)\}$
$=$any$(\{0: \mid x=y\}\cup\{1+i: \mid x\neq y\})$
where $i=$any(Search(x, r))
$=0$ if $x=y$
$1+i$ if $x\neq y$
where $i=$search(x, r)
$=0$ if $x=y$
$1+$search(x, r) if $x\neq y$

例4.4 商と余り

div$(x, y \mid x<y)$
$=$any(Div(x, y))
$=$any$(\{\langle 0, x\rangle\})$
$=\langle 0, x\rangle$
div$(x, y \mid x\geqq y)$
$=$any(Div(x, y))
$=$any$(\{\langle 1+q, r\rangle : \langle q, r\rangle \in \mathrm{Div}(x-y, y)\})$
$=$any$(\{\langle 1+q, r\rangle\})$
where$\langle q, r\rangle=$any(Div$(x-y, y)$)
$=\langle 1+q, r\rangle$
where$(q, r)=$div$(x-y, y)$

構造帰納法に基づくアルゴリズムの発見法の最も大きな特徴は，集合論仕様から，ある特定のアルゴリズムだけではなく，様々なアルゴリズムを系統的に発見することができることであろう．これによって，同一の仕様を満足する各種のアルゴリズムに共通する一般的な性質や相互の関連などを分析することが可能になる[8]．

5. 最適化に基づくアルゴリズムの発見

本章では，効率の良い再帰的プログラムを導出するための具体的方法として，最適化に基づくアルゴリズムの発見法を説明する[9]．

5.1 問 題 点

構造帰納法に基づくアルゴリズムの発見法によれば，再帰性のパターンに沿って集合論仕様を等価変換していくだけで，多くのアルゴリズムをある程度機械的に発見することができる．しかし，このようにして発見されたアルゴリズムが望ましい（効率の良い）アルゴリズムであるとは限らない．たとえば，最長上昇列問題の集合論仕様（例3.4）に右端逐次入力帰納法を適用すると，次のような再帰的プログラムが得られる．

lupseq$([])=0$
lupseq$(r\|[x])$
$=$max$(\{$lupseq(r), lpostupseq$(r, x)\})$
lpostupseq$([], x)=1$
lpostupseq$(r\|[y], x)$
$=$lpostupseq(r, x) if $y>x$
max(lpostupseq(r, x),
lpostupseq$(r, y)+1$) if $y\leqq x$

このアルゴリズムは，単純で分かりやすいが，$O(2^n)$であり，よく知られた$O(n\cdot\log n)$のアルゴリズム（ダイクストラのアルゴリズム）と比較すると，大変効率が悪い．

構造帰納法に基づくアルゴリズムの発見法をよく見直してみると，次のような問題点に気づく．

(1) 主ルーチンとサブルーチンがそれぞれ独立に導出されるので，制御の無駄や重複計算が発生する可能性がある．

(2) 再帰性のパターンが必ず入出力データの再帰的構造に基づいているので，不必要なデータの処理が発生する可能性がある．

したがって，構造帰納法だけでは効率の良いアルゴリズムの発見は困難であり，さらに強力な方法が必要になる．

5.2 最適化

複雑な（効率の良い）アルゴリズムの再帰性のパターンを分析してみると，基本的な再帰性のパターンを最適化したものになっていることが多いことが分かる．

再帰性＝再帰性＋最適化

すなわち，単純なアルゴリズムの再帰性のパターンを最適化すれば，複雑なアルゴリズムの再帰性のパターンが得られる．**4.2**で述べた再帰性の表現とこの表現を統合すれば，次のような表現になる．

再帰性＝構造帰納法＋最適化．．．＋最適化

すなわち，複雑な再帰性のパターンを導入するには，まず，構造帰納法により単純な再帰性のパターンを導入し，次に，必要に応じて，最適化を繰り返せばよい．

最適化に基づくアルゴリズムの発見法では，構造帰納法の適用によって生じた冗長性を排除するための少数の基本的な最適化手法を用意し，これらを組み合せることによって，複雑なアルゴリズムを発見する．たとえば，前項で得られた最長上昇列アルゴリズムに「履歴記憶」という最適化手法を適用すると，次のような$O(n^2)$のアルゴリズムが得られる．

$$\begin{aligned}
&\mathrm{lupseq}(r) = a \text{ where} \langle a, m\rangle = \mathrm{lupseq}^*(r)\\
&\mathrm{lupseq}^*([\,]) = \langle 0, [\,]\rangle\\
&\mathrm{lupseq}^*(r \| [x]) = \langle \max(\{a, b\}), m \| [b]\rangle\\
&\quad \text{where} \langle a, m\rangle = \mathrm{lupseq}^*(r),\\
&\qquad\qquad b = \mathrm{lpostupseq}(r, x, m)\\
&\mathrm{lpostupseq}([\,], x, [\,]) = 1\\
&\mathrm{lpostupseq}(r \| [y], x, m \| [b])\\
&\qquad = \mathrm{lpostupseq}(r, x, m) \quad \text{if } y > x\\
&\qquad \max(\{\mathrm{lpostupseq}(r, x, m), b+1\})\\
&\qquad\qquad \text{if } y \leqq x
\end{aligned}$$

さらにいくつかの最適化手法を適用すると，最終的に，ダイクストラのアルゴリズムが得られるが，説明は省略する[9]．

6. おわりに

本稿では，形式的仕様からプログラムを合理的に導出することを目指した集合論プログラミングという新しいプログラミングパラダイムを概観した．集合論プログラミングの基本的な考えは，きわめて単純である．すなわち，与えられた問題に対する集合論仕様を階層的に記述し，構造帰納法や最適化に基づくアルゴリズムの発見法によって，これを系統的に再帰的プログラムに変換すればよい．本稿では，理解を容易にするために，比較的簡単な例しか取り上げなかったが，本手法によれば，標準的な教科書[10]で扱われている大部分のアルゴリズムを統一的な枠組で導出することができる．

近年，プログラミング作業の合理化を目的として，種々の形式的仕様記述言語が提案され，実際に使われている[2),3)]．しかし，これらの言語をより有効なものにするためには，仕様記述法だけでなく，仕様からプログラムへの系統的な変換法を確立する必要がある．今後，集合論プログラミングのような形式化されたプログラミング手法がますます重要になるであろう．

参考文献

1) 野木兼六，中所武司：プログラミングツール，昭晃堂 (1989).
2) George, S., Haff, P., Haveland, K., Haxthar-sen, A. E., Milne, R., Nielsen, C. B., Prehn, S. and Wagner, K. R.: The RAISE Specification Language, Prentice-Hall (1992).
3) Spivey, J. M.: Understanding Z, Cambridge University Press (1988).
4) 野木兼六：集合論プログラミング，日本ソフトウェア科学会第6回大会予稿集 (1989).
5) 野木兼六：構造帰納法に基づくアルゴリズムの発見，コンピュータソフトウェア，Vol. 7, No. 4, pp. 39-59 (1990).
6) Manna, Z.: Mathematical Theory of Computation, McGrow-Hill Inc. (1974). 五十嵐滋(訳)：プログラムの理論，日本コンピュータ協会 (1975).

7) Burstall, R. M. and Darlington, J.: A Transformation System for Developing Recursive Programs, JACM, Vol. 24, No. 1, pp. 44-67 (1977).
8) 野木兼六：ソーティングアルゴリズムの新しい分類，コンピュータソフトウェア，Vol. 9, No. 4, pp. 53-72 (1992).
9) 野木兼六：最適化に基づくアルゴリズムの発見，コンピュータソフトウェア，Vol. 11, No. 4, pp. 20-43 (1994).
10) Aho, A. V., Hopcroft, J. E. and Ullman, J. D.: Data Structures and Algorithms, Addison-Wesley Publishing Company, Inc. (1983). 大野義夫(訳)：データ構造とアルゴリズム，培風館 (1987).

(平成7年4月17日受付)

野木 兼六（正会員）

1944年生．1967年東京工業大学理工学部卒業．1969年同大学院修士課程(数学専攻)修了．同年(株)日立製作所入社．現在，同社基礎研究所主任研究員．工学博士．プログラミング言語，ソフトウェア工学，プログラム理論などに興味を持つ．1987年大河内記念技術賞受賞．著書「プログラミングツール」(共著，昭晃堂)．ソフトウェア科学会，ACM，IEEE各会員．

付 録

標準的なデータ型と関数の意味を以下に示す．

（1） 集合

Set(a) aの有限部分集合の全体
any(a) 空でない集合aの任意の要素

（2） 数値

Nat 自然数の全体
Int 整数の全体
$x..y$ xからyまでの区間
$x..<y$ xから$y-1$までの区間

（3） 直積

$a \times b$ aとbの直積
$\langle x, y \rangle$ 直積の要素

（4） 列

Seq(k, a) a上の長さkの列の全体
Seq(a) a上の有限長の列の全体
$|r|$ 列rの長さ
dom(r) 列rの定義域
rng(r) 列rの値域
[] 空列
[x] 長さ1の列
$r \| s$ 列rとsの連結