漸次的組化と融合による関数プログラムの最適化

岩崎 英哉　胡 振江　武市 正人

本論文は，プログラムの最適化に基づくソフトウェア発展を追究するという立場から，融合 (Fusion) と組化 (Tupling) という独立したプログラム変換手法を組み合わせて，構成的アルゴリズム論の考えに基づく運算 (Calculation) を進める方法を提示する．融合は，関数合成をひとつにまとめることによって二つの関数間で受け渡される中間的データ構造の生成を抑制し，組化は，複数の関数を組にすることによって同一のデータ構造を辿るという制御の無駄を軽減する．提案する手法では，対象とする関数を関連関数と組化して Catamorphism と呼ばれる標準的な再帰パターン形を導出した後，Catamorphism に対する融合定理を繰返し適用する．その際必要に応じて，融合に成功するような関数を漸次的に創り出した上で組化変換を行うことによって，明白ではあるが効率のよくないプログラムを効率のよいものへ系統的に変換できる．本論文では，自明ではないアルゴリズムの具体的な導出例として，一・二次元最大部分和・積問題に本手法を適用し，その有効性を示す．

1 はじめに

関数プログラムでは，基本的な機能を持つ小さな部品関数を関数合成によって組み合わせて，大きなプログラム (関数) を記述する．このようなプログラムは簡潔で読みやすいが，そのままでは必ずしも効率のよいものになるとは限らない．その大きな原因は，関数合成間で中間的なデータが生成・消費される可能性があること，同一のデータ構造を複数の関数が辿る可能性があることにある．融合 (Fusion) 変換 [5] [24]と組化 (Tupling) 変換 [6] [13]は，このような問題を解決し関数プログラムの実行効率を改善するための，代表的なプログラム変換手法である．

最近，構成的アルゴリズム論 [8] [15] [17] [18]の立場からこれらのプログラム変換に取り組もうという研究が，注目を集めている．構成的アルゴリズム論の原形は Bird と Meertens によるリストに関する理論の定式化 (Bird–Meertens Formalism [3]，略して BMF) に見ることができる．その後，木などの一般的な再帰データ型とその上の関数をそれぞれ対象 (Object)・射 (Arrow) とするカテゴリ [11]として考え，少数の強力な変換定理を用いてプログラムを変換していく手法へと，構成的アルゴリズム論は発展していった．プログラム変換に対するこのようなアプローチは，プログラムに対するある種の “計算” と捉えることができる．以後，本論文ではこのような計算を“プログラム運算 (Program Calculation)” あるいは単に “運算” と呼ぶ．

運算において変換規則を適用するための一般的なアルゴリズム (すなわち運算を進めるアルゴリズム) の構築，個々の標準関数 (*map*，*foldl* など) に対する変換規則の自動導出は構成的アルゴリズム論では扱われず，ある種の運算は，変換規則を既知のものとして人間が紙の上で行うのが現状である．しかしながら，構成的アルゴリズ

Calculating Fucntional Programs by Incremental Tupling and Fusion.

Hideya Iwasaki, Zhenjiang Hu, Masato Takeichi, 東京大学 大学院工学系研究科 情報工学 / 計数工学専攻, Department of Information Engineering / Mathematical Engineering, University of Tokyo.

コンピュータソフトウェア, Vol.18, No.0(2001), pp.46–59.
[論文] 2000 年 1 月 20 日受付.

ム論は，次のような大きな特徴を持っている．

- 自明ではあるが効率のよくないプログラムの記述からの効率のよいプログラム (アルゴリズム) の導出というプログラム発展の過程を，運算という枠組で捉えることが可能である．このことは，効率のよい既知のアルゴリズムの運算による導出，ひいては，未知の新しいアルゴリズムを発見 (導出) する可能性へとつながる．
- 構成的アルゴリズム論の理論には，foldr/build 則 [10] [16]，酸性雨定理 [23]，Hylomorphism と呼ばれる再帰関数パターンの導出 [12]のように，発見的ではない機械的な適用が可能なものもある．このような自動化可能な変換定理に基づき，自動変換システム [19] [20]を構築することが可能である．

本論文は，前者に注目するという立場に立ち，独立した二つの変換手法 — 融合と組化 — を組み合わせて運算を進める際の指針を提案する．さらに，自明ではないアルゴリズムの具体的な導出例を示し，提案する手法の有効性を立証する．

本論文で提案する手法には，次のような特徴がある．

- 本手法は，構成的アルゴリズム論における代表的な再帰パターンのひとつである Catamorphism を組化によって導入し，Catamorphism に対する融合変換定理に基づいて変換を進める．そのため，融合変換定理が適用可能となるための条件を考慮することによる，明確な手順に基づく系統的なプログラム変換が可能となる．
- 本手法では，融合変換定理の適用可能条件の考慮の際，必要に応じて組化変換を行い，融合が成功するような新しい関数を漸次的に創り出す．

本論文の構成は次の通り．第 2 節で構成的アルゴリズム論を概説し，本論文で必要となる融合定理，組化定理に関して簡単に説明する．続く第 3 節では，我々の提案する手法の基本部分を説明し，簡単な例題および一次元最大部分列和問題に適用する．第 4 節では，第 3 節までの議論の単純な適用では対処できないような問題に対するため，組化変換を漸次的に施す手法を提案する．さらに，その適用例として，一次元最大部分列積問題をとりあげる．第 5 節では，自明ではない問題 (二次元最大部分列和・積問題) を例題とし，第 3，4 節で提案した手法の有効性を示す．最後に，第 6 節で関連研究について述べる．

$$
\begin{array}{ll}
\text{(D–}length\text{)} & length\,[\,] = 0 \\
 & length\,(x : xs) = 1 + sum\,xs \\
\text{(D–}sum\text{)} & sum\,[\,] = 0 \\
 & sum\,(x : xs) = x + sum\,xs \\
\text{(D }prod\text{)} & prod\,[\,] = 1 \\
 & prod\,(x : xs) = x * prod\,xs \\
\text{(D–}sum2\text{)} & sum2\,[\,] = 0 \\
 & sum2\,(xs : xss) = sum\,xs + sum2\,xss \\
\text{(D }prod2\text{)} & prod2\,[\,] = 1 \\
 & prod2\,(xs : xss) = prod\,xs * prod2\,xss \\
\text{(D }max\text{)} & max\,[\,] = -\infty \\
 & max\,(x : xs) = x \uparrow max\,xs \\
\text{(D–}min\text{)} & min\,[\,] = \infty \\
 & min\,(x : xs) = x \downarrow min\,xs \\
\text{(D }map\text{)} & map\,f\,[\,] = [\,] \\
 & map\,f\,(x : xs) = f\,x : map\,f\,xs \\
\text{(D–}inits\text{)} & inits\,[\,] = [[\,]] \\
 & inits(x : xs) = [\,] : map(x :)(initsxs)
\end{array}
$$

図 1　本論文の運算で用いる標準関数

本論文では，関数プログラム的記法を用いてプログラムを記述し，標準的な関数は [4]にあるものを用いる．ここで，特に注意すべき点について説明しておく．

- 関数適用は，関数と引数との間に空白を入れることによって示す．
- 関数はカリー化して記述し，引数は左に結合するものとする．したがって，$f\,a\,b$ は $(f\,a)\,b$ を意味する．
- 関数適用は，他の演算子と比較して，最も結合の優先順位が強いものとみなす．
- 関数合成は $\circ$ で表す．すなわち，$(f \circ g)x = f(gx)$ である．
- 二項演算子 $\oplus$ は，$(a \oplus)$，$(\oplus b)$，$(\ominus)$ のように括弧で囲む ("セクション化" と呼ぶ) ことにより関数とすることができる．すなわち，

$$a \oplus b = (a \oplus)\,b = (\oplus b)\,a = (\ominus)\,(a, b)$$

(map–$append$ 則)	$map\ f\ (xs +\!\!+ ys) = map\ f\ xs +\!\!+ map\ f\ ys$
(map $concat$ 則)	$map\ f\ (concat\ xss) = concat\ (map\ (map\ f)\ xss)$
(map 分配則)	$map\ f\ (map\ g\ xs) = map\ (f \circ g)\ xs$
(max–$append$ 則)	$max\ (xs +\!\!+ ys) = max\ xs \uparrow max\ ys$
(max–$concat$ 則)	$max\ (concat\ xss) = max\ (map\ max)\ xss)$
(max–加算則)	$max\ (map\ (x+)\ xs) = x + max\ xs$
(max–乗算則)	$max\ (map\ (x*)\ xs) =$ if $x \geq 0$ then $x * max\ xs$ else $x * min\ xs$
(min–乗算則)	$min\ (map\ (x*)\ xs) =$ if $x \geq 0$ then $x * min\ xs$ else $x * max\ xs$
(map–$zipwith$ 則)	$map\ f\ (zipwith\ g\ xs\ ys) = zipwith\ (f \circ g)\ xs\ ys$
($zipwith$–map 則)	$zipwith\ (f \circ (g \times h))\ xs\ ys = zipwith\ f\ (map\ g\ xs)\ (map\ h\ ys)$

図 2　本論文で用いる変換規則

- 組を $(\ ,\ )$ という記法で表し，組に関連する演算子を下のように定義する．

  $(f \triangle g)\ x = (f\ x, g\ x)$

  $(f \times g)\ (x, y) = (f\ x, g\ y)$

図 1 には，本論文で用いる標準的な関数のうち運算で必要なものの定義を，図 2 には，個々の標準関数に関連する変換規則のうち，本論文で用いるものをまとめて示す．ただし，$\uparrow$ と $\downarrow$ はそれぞれオペランドのうち大きい値と小さい値を返す二項演算子を表す．先にも述べた通り，図 2 の規則は既知のものと仮定する．また，本論文に示す運算過程では，利用した規則等を等号直後の $\{\ \}$ の中に示す．

2　構成的アルゴリズム論とプログラム変換

本論文で提案する手法は，木などの一般的な再帰データ型を扱う関数についても適用できる汎用的な方法であるが，本論文では簡単のため，データ構造をリスト型に限定して議論し，各種変換定理もリスト型に特化した形で提示する．

2.1　融合変換と組化変換

融合変換は関数合成をひとつの関数にまとめることにより，中間データの生成・消費を抑制する．たとえば，リストの各要素に，与えられた関数を適用した結果をリストとして返す関数 map と，リストの構成要素の和を求める関数 sum を考える．数を 2 倍する関数を $(2*)$ と記述すると，$sm\ xs = sum\ (map\ (2*)\ xs)$ は，xs の各要素の 2 倍の和を返す関数になる．ところがこのままでは，map によって中間的なリストが過渡的に生成され，最終結果には出現せずに sum によって消費される．融合変換は，このような関数合成をひとつにまとめ，sm の定義を次のように変換する．

$sm\ [\,] = 0$

$sm\ (x : xs) = 2 * x + sm\ xs$

これによって，中間的リスト構造が除去され，時間・空間の両面において実行効率が大きく改善される．

一方，組化は同一の実引数 (データ構造) を持つ関数を組にして，そのデータ構造を複数の関数が別個に辿ることによる制御の重複を除く．たとえば，次のような関数 $average$ を考える．

$average\ xs = sum\ xs\ /\ length\ xs$

$average$ は，数のリストの要素の平均値を求める関数であるが，同一のリストを sum と $length$ がそれぞれ別個に辿ることによる非効率性を内在している．sum と $length$ を組にして組化変換を行うと，

$average\ xs = u/v$ where $(u, v) = sumlen\ xs$

$sumlen\ [\,] = (0, 0)$

$sumlen\ (x : xs) = (x + u,\ 1 + v)$

where $(u, v) = sumlen\ xs$

となる．sum と $length$ の定義をひとつにまとめた関数 $sumlen$ により，リストを辿る回数が 1 回ですむようになり，非効率性が解消された．

2.2 融合定理と組化定理

リストを消費する自然な再帰関数 f は，ϕ を定数，ψ を適当な関数として，次のように定義される．

$$f\,[\,] = \phi$$
$$f\,(x : xs) = \psi\,(x,\, f\,xs)$$

ϕ，ψ を定めれば f は一意に定まるので，この f を $(\!|\phi, \psi|\!)$ と表記[†1]する [15] [18]．たとえば関数 sum は，$(\!|0, (+)|\!)$ と表現される．このような自然な再帰形で表現される関数一般を，リスト上の Catamorphism と呼ぶ．以後，Catamorphism を単に Cata と略記する．

Cata に対する融合変換の基本が，次の融合定理 [17] [18]である．この定理は，ある関数 h と Cata との合成が別の Cata となるための条件を示している．

定理 1 (Cata 融合定理)

$$h\,\phi = \phi',\quad h\,(\psi\,(x, r)) = \psi'\,(x, h\,r)$$
$$\Longrightarrow h \circ (\!|\phi, \psi|\!) = (\!|\phi', \psi'|\!) \qquad \square$$

本定理の適用によって，h と $(\!|\phi, \psi|\!)$ の間の関数合成をひとつに融合することが可能であり，これらの関数間で渡される (中間的) データ構造を生成しないようなプログラムに変換できる．

第 2.1 節であげた例 $sum \circ map\,(2*)$ にこの定理を適用してみよう．定義より $map\,(2*)$ は $(\!|[\,], \psi_{map}|\!)$，ただし，

$$\psi_{map}\,(x, r) = 2 * x : r$$

という Cata で表現される．sum との合成を融合するために，上記融合定理の前提部を考える．$sum\,[\,] = 0$ は自明なので，次に，

$$sum\,(\psi_{map}\,(x, r)) = \psi'_{map}\,(x,\, sum\,r)$$

を満たす ψ'_{map} を求める．これは，次のような簡単な運算によって定めることができる．

$$\begin{aligned}
&sum\,(\psi_{map}\,(x, r))\\
&= \{\ \psi_{map}\text{の定義}\ \}\\
&\quad sum\,(2 * x : r)\\
&= \{\ sum\text{の定義}\ \}\\
&\quad 2 * x + sum\,r\\
&= \{\ \text{定理の前提条件}\ \}\\
&\quad \psi'_{map}\,(x,\, sum\,r)
\end{aligned}$$

したがって，

$$\psi'_{map}\,(x, s) = 2 * x + s$$

と決定され，融合定理より，

$$sum \circ map\,(2*) = (\!|0, \psi'_{map}|\!)$$

と融合することに成功した．最後に上式の Cata を通常の定義形に形式的に書きかえて，第 2.1 節にあげた sm の定義が得られる．

もうひとつの重要な定理が，組化変換を行う組化定理 (Mutu Tupling Theorem) [7] [8] である．組化変換では，同一のリスト構造を辿る (制御に重複のある) 複数の関数を組にしてひとつの関数にまとめて，制御の重複を除去する．ここで組にする関数は，独立した関数でもよいし，相互再帰などの依存関係があってもよい．

定理 2 (組化定理)

$$f\,[\,] = \phi_f,\quad f\,(x : xs) = \psi_f\,(x,\, (f \vartriangle g)\,xs)$$
$$g\,[\,] = \phi_g,\quad g\,(x : xs) = \psi_g\,(x,\, (f \vartriangle g)\,xs)$$
$$\Longrightarrow f \vartriangle g = (\!|(\phi_f, \phi_g),\, \psi_f \vartriangle \psi_g|\!) \qquad \square$$

この定理の適用例として，次の関数 $bign$ を考える．

$$bign\,[\,] = [\,]$$
$$bign\,(x : xs)$$
$$= \mathsf{if}\ x > sum\,xs\ \mathsf{then}\ x : bign\,xs\ \mathsf{else}\ bign\,xs$$

この関数は，リストの中から，自分より後の要素の和よりも大きな値を持つ要素だけを抽出したリストを返す．$bign$ は sum に依存し，これらは同一のデータ構造を辿るので，このままでは制御の無駄が生じ，リストの長さを N とすると $O(N^2)$ の手間を必要とする．

$bign$ と sum は同じデータを引数とするので，

$$\psi_{bign}\,(x, (ys, s)) = \mathsf{if}\ x > s\ \mathsf{then}\ x : ys\ \mathsf{else}\ ys$$
$$\psi_{sum}\,(x, (ys, s)) = x + s$$

と定義すると，組化定理により

$$bign \vartriangle sum = (\!|([\,], 0),\, \psi_{bign} \vartriangle \psi_{sum}|\!)$$

となる．この Cata を F とおいて通常の定義形に書き直すと，次のようになる．

$$F\,[\,] = ([\,], 0)$$
$$F\,(x : xs) = (\mathsf{if}\ x > s\ \mathsf{then}\ x : ys\ \mathsf{else}\ ys,\, x + s)$$
$$\mathsf{where}\ (ys, s) = F\,xs$$

F は，引数のリストを 1 回しか辿らず，制御の無駄が解消されて $O(N)$ のプログラムに改善されている．$bign$ は F の返す組の第一要素を抽出すればよい．

$$bign\,xs = u\ \mathsf{where}\ (u, v) = F\,xs$$

組化の効用は，制御の無駄を軽減するだけにはとどま

†1 [15]では $(\!|\phi \triangledown \psi|\!)$ と表記しているが，本論文はリストに限定して議論するので，このように表記する．

らない．上の例でも見られる通り，単独では Cata でない関数 ($bign$) であっても，別の関数 (sum) と組化することによって，組化によって生まれた新しい関数 (F) は Cata として記述されるようになる．その結果，構成的アルゴリズム論の枠組を用いた変換が可能となる．この点は組化定理の重要なポイントである．

組化によって生まれた Cata に関する融合定理は，先に示した定理の特殊な場合である．

系 3 (組化 Cata 融合定理)

$$
\begin{aligned}
&h_1\,\phi_f = \phi'_f\\
&h_1\,(\psi_f\,(x,(s_1,s_2))) = \psi'_f\,(x,(h_1\,s_1,h_2\,s_2))\\
&h_2\,\phi_g = \phi'_g\\
&h_2\,(\psi_g\,(x,(s_1,s_2))) = \psi'_g\,(x,(h_1\,s_1,h_2\,s_2))\\
&\Longrightarrow (h_1\times h_2)\circ(\!|(\phi_f,\phi_g),\psi_f\vartriangle\psi_g|\!)\\
&\qquad = (\!|(\phi'_f,\phi'_g),\psi'_f\vartriangle\psi'_g|\!)
\end{aligned}
$$

□

関数合成 $h\circ f$ を融合する際，f と他の関数 g との組 $f\vartriangle g$ が Cata として表現されるならば，上の組化 Cata 融合定理を適用する必要が生じる．その際，h_1 は h とすればよいが，h_2 としては定理の前提条件を満足するように適切に定める必要がある．

3 組化 Catamorphism に対する融合

融合と組化は独立したプログラム変換手法なので，運算を進めるにあたって，これらをどのように組み合わせて適用すればよいかに関する指針を与えることが望ましい．本論文は，$h\circ f$ という関数合成に対して，f に対して (f と組化可能な関数と共に) 組化定理を適用して Cata を導出した上で組化 Cata 融合定理を適用し，構成的アルゴリズム論の枠組でのプログラム変換を可能にするという方法を提案する．本方法の手順は次のようにまとめられる．ただし簡単のため，ステップ 1 において，f と組化する関数は g だけで，大きさ 2 の組が作られるとした．3 個以上の関数を組化する場合の手順も，全く同様である．

1. f と組化可能な関数を g とする．これらについて組化定理を適用し，$f\vartriangle g$ に対する Cata 表現 $(\!|(\phi_f,\phi_g),\psi_f\vartriangle\psi_g|\!)$ を導出する．
2. $(h\times h_2)\circ(f\vartriangle g)$ に対して組化 Cata 融合定理が適用可能となるように関数 h_2 を定め，融合操作を行う．
3. 前ステップで得られた (融合後の) 関数の返す結果の第一要素を取り出す．

複数個の関数合成がある場合には，融合すべき関数合成の個数回だけ上記ステップ 2 を繰り返す．

上の手順の具体的適用例として，$lb = length\circ bign$ の融合変換を試みる．前節で示した通り，$bign$ は sum と組化され $(\!|([],0),\psi_{bign}\vartriangle\psi_{sum}|\!)$ という Cata で表現された．次に適当な関数 h_2 を導入して，$(length\times h_2)\circ(bign\vartriangle sum)$ に対して組化 Cata 融合定理を適用する．定理の前提条件を考慮すると，

$$
\begin{aligned}
&length\,[\,] = 0\\
&length\,(\psi_{bign}\,(x,(ys,s)))\\
&= \{\ \psi_{bign}\text{の定義}\ \}\\
&\quad length\,(\mathsf{if}\ x>s\ \mathsf{then}\ x:ys\ \mathsf{else}\ ys)\\
&= \{\ length\text{を if の中に移動，}length\text{の定義}\ \}\\
&\quad \mathsf{if}\ x>s\ \mathsf{then}\ 1+length\,ys\ \mathsf{else}\ length\,ys\\
&= \{\ \text{定理の前提条件}\ \}\\
&\quad \psi'_{bign}\,(x,(length\,ys,h_2\,s))
\end{aligned}
$$

なので，id を恒等関数として

$$
\begin{aligned}
&h_2 = id\\
&\psi'_{bign}\,(x,(n,s)) = \mathsf{if}\ x>s\ \mathsf{then}\ 1+n\ \mathsf{else}\ n
\end{aligned}
$$

と定めれば，$h_2\,0 = 0$，$h_2\circ\psi_{sum} = \psi_{sum}$ より，

$$
\begin{aligned}
&(length\times id)\circ(bign\vartriangle sum)\\
&\qquad = (\!|(0,0),\psi'_{bign}\vartriangle\psi_{sum}|\!)
\end{aligned}
$$

と融合される．この関数を F'_{lb} とおいて通常の形に記述しなおすと，次のような結果が得られる．

$$
\begin{aligned}
&lb\,xs = u\ \mathsf{where}\ (u,v) = F'_{lb}\,xs\\
&F'_{lb}\,[\,] = (0,0)\\
&F'_{lb}\,(x:xs) = (\mathsf{if}\ x>s\ \mathsf{then}\ 1+n\ \mathsf{else}\ n,\ x+s)\\
&\quad \mathsf{where}\ (n,s) = F'_{lb}\,xs
\end{aligned}
$$

上記ステップ 1 ～ 3 に基づく組化定理と融合定理の連続適用により，同一のデータ構造は一度しか辿らず中間データ ($bign$ の結果) も生成しない，効率のよいプログラムへと変換された．

別の例として，最大部分列和 (Maximum Segment Sum，略して MSS) 問題に対して，本方法を適用する．この問題は，データマイニングにおける最大利得範囲 (Maximum Gain Range) 問題 [9]と同等であることが知られており，実用的な意味でも重要な問題である．

MSS 問題とは，整数の有限列 (リスト) xs の中の連続

する部分列の要素の和の最大値を求める問題である．ただし，長さが0の空部分列の要素の和は0とする．たとえば，$xs = [3, -4, 2, -1, 6, -3]$ の MSS は，部分列 $[2, -1, 6]$ の和の 7 である．この問題に関して，素朴で明白であるが効率のよくない初期プログラムからの効率のよいプログラム導出は，プログラム変換の分野においてしばしば用いられる例題であり，今までにも [3] に示されるような手法が提案されている．

MSS 問題の素朴な初期プログラム $mss\,xs$ は，xs のすべての部分列を列挙したリストを作る関数 $segs$，部分列の要素の和を求める関数 sum，リストの要素の最大値を求める関数 max を用いた関数合成により，次のように記述できる．

$$mss = max \circ map\,sum \circ segs$$
$$segs\,[\,] = [[\,]]$$
$$segs\,(x : xs) = inits\,(x : xs) \mathbin{+\!\!+} segs\,xs$$

さて，$segs$ はあらゆる部分列を列挙した長い (中間) リストを生成し，次に $map\,sum$ はこれと同じ長さの別の中間リストを生成する．したがって，プログラムの効率改善のためには，$max \circ map\,sum \circ segs$ の融合が必要である．上の定義から明らかな通り，$segs$ は単独では Cata ではないが，$inits$ と同じ引数を共有するので，$segs$ と $inits$ に対して組化定理を用いて組化変換する．$F_{mss} = segs \vartriangle inits$ とおき結果のみを示すと，次のようになる．

$$F_{mss} = (\!|\,([[\,]], [[\,]]),\ \psi_{segs} \vartriangle \psi_{inits}\,|\!)$$
$$\psi_{segs}\,(x, (yss, zss)) = ([\,] : map\,(x :)\,zss) \mathbin{+\!\!+} yss$$
$$\psi_{inits}\,(x, (yss, zss)) = [\,] : map\,(x :)\,zss$$

次に，関数合成 $map\,sum \circ segs$ を融合するために，適当な関数 h_2 について $(map\,sum \times h_2) \circ F_{mss}$ の融合を考える．まず，組化 Cata 融合定理の前提条件の一方，

$$map\,sum\,(\psi_{segs}\,(x, (yss, zss))) = \psi'_{segs}\,(x, (map\,sum\,yss, h_2\,zss))$$

を満たす ψ'_{segs} と h_2 を決定する．

$$\begin{aligned}
&map\,sum\,(\psi_{segs}\,(x, (yss, zss)))\\
&= \{\ \psi_{segs}\text{の定義}\ \}\\
&\quad map\,sum\,(([\,] : map\,(x :)\,zss) \mathbin{+\!\!+} yss)\\
&= \{\ map\text{-}append\text{則}，\ map\text{の定義}，\ sum\text{の定義}\ \}\\
&\quad (0 : map\,sum\,(map\,(x :)\,zss)) \mathbin{+\!\!+} map\,sum\,yss\\
&= \{\ map\text{分配則}\ \}\\
&\quad (0 : map\,(sum \circ (x :))\,zss) \mathbin{+\!\!+} map\,sum\,yss\\
&= \{\ sum\text{の定義}\ \}\\
&\quad (0 : map\,((x +) \circ sum)\,zss) \mathbin{+\!\!+} map\,sum\,yss\\
&= \{\ map\text{分配則}\ \}\\
&\quad (0 : map\,(x +)\,(map\,sum\,zss)) \mathbin{+\!\!+} map\,sum\,yss\\
&= \{\ \text{定理の前提条件}\ \}\\
&\quad \psi'_{segs}\,(x, (map\,sum\,yss, h_2\,zss))
\end{aligned}$$

なので，

$$h_2 = map\,sum$$
$$\psi'_{segs}\,(x, (ys, zs)) = (0 : map\,(x +)\,zs) \mathbin{+\!\!+} ys$$

と定めればよい．同様に，もうひとつの前提条件

$$\begin{aligned}
&h_2\,(\psi_{inits}\,(x, (yss, zss)))\\
&\quad = \psi'_{inits}\,(x, (map\,sum\,yss, h_2\,zss))
\end{aligned}$$

から，

$$\psi'_{inits}\,(x, (ys, zs)) = 0 : map\,(x +)\,zs$$

と定められる．また，$map\,sum\,[[\,]] = [0]$ であるから，$(map\,sum \times h_2) \circ F_{mss}$ は $(\!|\,([0], [0]),\ \psi'_{segs} \vartriangle \psi'_{inits}\,|\!)$ に融合された．この融合結果の Cata を F'_{mss} とおく．

次に，max との関数合成を融合するために，適当な関数 h'_2 を導入し，$(max \times h'_2) \circ F'_{mss}$ を融合する．前と同様にして定理の前提条件

$$\begin{aligned}
&max\,(\psi'_{segs}\,(x, (ys, zs)))\\
&\quad = \psi''_{segs}\,(x, (max\,ys, h'_2\,zs))
\end{aligned}$$

について運算を行うと，

$$\begin{aligned}
&max\,(\psi'_{segs}\,(x, (ys, zs)))\\
&= \{\ \psi'_{segs}\text{の定義}\ \}\\
&\quad max\,((0 : map\,(x +)\,zs) \mathbin{+\!\!+} ys)\\
&= \{\ max\text{-}append\text{則}，\ max\text{の定義}\ \}\\
&\quad 0 \uparrow max\,(map\,(x +)\,zs) \uparrow max\,ys\\
&= \{\ max\text{-加算則}\ \}\\
&\quad 0 \uparrow (x + max\,zs) \uparrow max\,ys\\
&= \{\ \text{定理の前提条件}\ \}\\
&\quad \psi''_{segs}\,(x, (max\,ys, h'_2\,zs))
\end{aligned}$$

なので，$h'_2 = max$ として

$$\psi''_{segs}\,(x, (y, z)) = 0 \uparrow (x + z) \uparrow y$$

と定められる．同様にして他方の前提条件

$$\begin{aligned}
&h'_2\,(\psi'_{inits}\,(x, (ys, zs)))\\
&\quad = \psi''_{inits}\,(x, (sum\,ys, h'_2\,zs))
\end{aligned}$$

から，

$$\psi''_{inits}\,(x,\,(y,z)) = 0 \uparrow (x+z)$$

となる．以上と $max\,[0] = 0$ から，$(max \times h'_2) \circ F'_{mss}$ を

$$F''_{mss} = (\!|\,(0,0),\,\psi''_{segs} \vartriangle \psi''_{inits}\,|\!)$$

と融合することに成功した．この最終結果を通常の定義の形で表現し，共通に現れる値 $0 \uparrow (x+z)$ をまとめると，次のようになる．

$$mss\;xs = y \;\text{ where }\; (y,z) = F''_{mss}\;xs$$
$$F''_{mss}\;[\,] = (0,0)$$
$$F''_{mss}\;(x : xs) = (w \uparrow y,\; w)$$
$$\text{where }\; (y,z) = F''_{mss}\;xs$$
$$w = 0 \uparrow (x+z)$$

入力リストの長さを N とすると，初めの mss は $O(N^3)$ の時間計算量を必要としたのに対し，上記運算により $O(N)$ のプログラムが得られた．これは，よく知られたアルゴリズム [1]，他の方法によって得られた最終プログラム [3]と実質的に等価であるが，我々の提案する手法に基づく系統的な変換により導出された．

4 漸次的組化

前節で述べた変換結果は満足のいくものであったが，このような変換がいつも順調にいくとは限らない．その理由は，$(h \times h_2) \circ (f \vartriangle g)$ に対する組化 Cata 融合定理の前提部

$$h\;(\psi_f\;(x,\,(s_1,s_2))) = \psi'_f\;(x,\,(h\;s_1,\,h_2\;s_2))$$

を満たすように h_2 と ψ'_f を決定する際，左辺の展開形の中に s_2 が (異なる関数の引数となって) 数ヶ所に分散して出現することがあるためである．本論文では，このような場合に s_2 を引数とする複数の関数を組化することにより，この問題を解決することを提案する．すなわち，上式の左辺の展開形が

$$h\;(\psi_f\;(x,\,(s_1,s_2))) = \eta\;(x,\,h\;s_1,\,h_{21}\;s_2,\,h_{22}\;s_2)$$

という形になった場合，右辺の h_{21}，h_{22} を組化して $h_{21} \vartriangle h_{22}$ として，

$$h\;(\psi_f\;(x,\,(s_1,s_2))) = \eta\;(x,\,h\;s_1,\,u,\,v)$$
$$\text{where }\; (u,v) = (h_{21} \vartriangle h_{22})\;s_2$$

と変形する．すると，h_2 と ψ'_f を次のように決定することができる．

$$h_2 = h_{21} \vartriangle h_{22}$$
$$\psi'_f\;(x,\,(y,\,(u,v))) = \eta\;(x,y,u,v)$$

以上をまとめると，前節のはじめに示した 3 ステップの中の 2 ステップ目は次のようになる．

2. $(h \times h_2) \circ (f \vartriangle g)$ に対して融合定理が適用できるような適当な関数 h_2 を導入し，融合操作を行う．その際，必要に応じて組化変換を行い h_2 となる関数を新たに創り出す．

この段階における組化を “漸次的組化” と呼ぶ．このように呼ぶ理由は，連続した関数合成を融合する際には，上のステップ 2 が繰返し適用され，そのたびに必要な組化が要求駆動的に行われるためである．

漸次的組化が必要となる具体例として，MSS 問題と類似した最大部分列積 (Maximum Segment Product，略して MSP) 問題を考える．MSP 問題とは，MSS 問題での “和” を “積” に交換した (ただし空部分列の積は 1 とする) ものである．MSS と MSP は，仕様の上では加算と乗算が異なるだけの大変よく似た問題だが，後述するような加算と乗算の性質の違いにより，プログラム変換は MSP 問題の方が格段に難しい．

リストの要素の積を求める関数を $prod$ とすれば，MSP の素朴な初期プログラム msp は mss と同様に

$$msp = max \circ map\;prod \circ segs$$

で与えられる．この問題に対して，MSS と同様の手順のプログラム変換を適用する．$segs$ と $inits$ を $F_{msp} = segs \vartriangle inits$ と組化した後，適当な関数 h_2 を導入し，$(map\;prod \times h_2) \circ F_{msp}$ の融合を行うまでは，前節の MSS 問題と全く同様にして進めることができる．実際，$h_2 = map\;prod$ と定めて次のような結果が得られる．

$$(map\;prod \times map\;prod) \circ F_{msp}$$
$$= (\!|\,([1],[1]),\,\psi'_{segs} \vartriangle \psi'_{inits}\,|\!)$$
$$\psi'_{segs}\;(x,\,(ys,zs)) = (1 : map\;(x*)\;zs) +\!\!+\; ys$$
$$\psi'_{inits}\;(x,\,(ys,zs)) = 1 : map\;(x*)\;zs$$

この融合結果の Cata を F'_{msp} とおく．次に，適当な関数 h'_2 を導入し，$(max \times h'_2) \circ F'_{msp}$ の融合を考える．はじめに Cata 融合定理の前提条件

$$max\;(\psi'_{segs}\;(x,\,(ys,zs)))$$
$$= \psi''_{segs}\;(x,\,(max\;ys,\,h'_2\;zs))$$

の左辺を MSS の場合と同様にして変形すると，

$$max\;(\psi'_{segs}\;(x,\,(ys,zs)))$$
$$= \{\;\psi'_{segs}\text{の定義}\;\}$$

$max\,(1 : map\,(x*)\,zs)\ +\!\!+\ ys$

$=$ { maxの定義，max–$append$則 }

$1 \uparrow max\,(map\,(x*)\,zs) \uparrow max\,ys$

$=$ { max–乗算則 }

$1 \uparrow (x * (\mathsf{if}\ x \geq 0\ \mathsf{then}\ max\,zs\ \mathsf{else}\ min\,zs))$
$\uparrow max\,ys$

となる．最後の式の通り，zs に対して $max zs$ と $min zs$ の二つの関数適用が出現する．MSS の場合は，これに対応する式には $max\,zs$ しか出現しなかった (したがって $h'_2 = max$ と即座に決定できた) ことに注意されたい．この違いは，MSS の場合は加算の“単調性” (図2の max–加算則) により，最大値をとる操作と x を加える操作の順番を逆にできたのに対し，MSP の場合は，負の値の乗算によって正負が逆転する (乗算は“単調”ではない) ために，最大値をとる操作と x の乗算を単純に逆にすることができず，最小値をも考慮に入れなければならないことに原因がある．

上の場合，同じ zs を引数とする max と min を漸次的組化し $(max \triangle min)\,zs$ として，$h'_2 = max \triangle min$ と決めることができる．すなわち，

上式

$=$ { $h'_2 = max \triangle min$ }

$1 \uparrow (x * (\mathsf{if}\ x \geq 0\ \mathsf{then}\ u\ \mathsf{else}\ v)) \uparrow max\,ys$
$\mathsf{where}\ (u, v) = h'_2\,zs$

$=$ { 定理の前提条件 }

$\psi''_{segs}\,(x, (max\,ys, h'_2\,zs))$

となるので，

$\psi''_{segs}\,(x, (y, (u, v)))$

$= 1 \uparrow (x * (\mathsf{if}\ x \geq 0\ \mathsf{then}\ u\ \mathsf{else}\ v)) \uparrow y$

と定められる．同様に，定理の他方の前提条件

$h'_2\,(\psi'_{inits}\,(x, (ys, zs)))$
$= \psi''_{inits}\,(x, (max\,ys, h'_2\,zs))$

から，ψ''_{inits} を決定する．左辺の値は組であり，その第一・第二の各要素に注目して運算すると，

第一要素

$= max\,(\psi'_{inits}\,(x, (ys, zs)))$

$=$ { ψ'_{inits}の定義 }

$max\,(1 : map\,(x*)\,zs)$

$=$ { maxの定義 }

$1 \uparrow max\,(map\,(x*)\,zs)$

$=$ { max–乗算則 }

$1 \uparrow (x * (\mathsf{if}\ x \geq 0\ \mathsf{then}\ max\,zs\ \mathsf{else}\ min\,zs))$

$=$ { $h'_2 = max \triangle min$ }

$1 \uparrow (x * (\mathsf{if}\ x \geq 0\ \mathsf{then}\ u\ \mathsf{else}\ v))$
$\mathsf{where}\ (u, v) = h'_2\,zs$

第二要素

$= min\,(\psi'_{inits}\,(x, (ys, zs)))$

$=$ { ψ'_{inits}の定義 }

$min\,(1 : map\,(x*)\,zs)$

$=$ { minの定義 }

$1 \downarrow min\,(map\,(x*)\,zs)$

$=$ { min–乗算則 }

$1 \downarrow (x * (\mathsf{if}\ x \geq 0\ \mathsf{then}\ min\,zs\ \mathsf{else}\ max\,zs))$

$=$ { $h'_2 = max \triangle min$ }

$1 \downarrow (x * (\mathsf{if}\ x \geq 0\ \mathsf{then}\ v\ \mathsf{else}\ u))$
$\mathsf{where}\ (u, v) = h'_2\,zs$

したがって，

$\psi''_{inits}\,(x, (y, (u, v)))$
$= (1 \uparrow (x * (\mathsf{if}\ x \geq 0\ \mathsf{then}\ u\ \mathsf{else}\ v)),$
$1 \downarrow (x * (\mathsf{if}\ x \geq 0\ \mathsf{then}\ v\ \mathsf{else}\ u)))$

が得られた．以上と $max\,[1] = 1$，$h'_2\,[1] = (1, 1)$ から，$(max \times h'_2) \circ F'_{msp}$ は

$F''_{msp} = (\!|\,(1, (1, 1)), \psi''_{segs} \triangle \psi''_{inits}\,|\!)$

と融合される．これを通常の形で表現し，条件部 $(x \geq 0)$ をくくり出して共通に現われる値をまとめると，次のような最終プログラムが得られる．

$msp\,xs = y\ \mathsf{where}\ (y, (u, v)) = F''_{msp}\,xs$

$F''_{msp}\,[\,] = (1, (1, 1))$

$F''_{msp}\,(x : xs) = \mathsf{if}\ x \geq 0\ \mathsf{then}\ (a \uparrow y, (a, 1 \downarrow q))$
$\mathsf{else}\ (b \uparrow y, (b, 1 \downarrow p))$

$\mathsf{where}\ (y, (u, v)) = F''_{msp}\,xs$
$p = x * u,\ \ a = 1 \uparrow p$
$q = x * v,\ \ b = 1 \uparrow q$

漸次的組化を利用した系統的運算により，リストの長さを N とした時，$O(N^3)$ の素朴な初期プログラムから $O(N)$ のプログラムを得ることに成功した．

5 二次元最大部分列和・積問題への応用

本節では，本論文で提案する手法を，より複雑な自明でない問題に適用し，その有効性を示す．ここでとりあ

げる二次元最大部分列和 (MSS2)・部分列積 (MSP2) 問題は，MSS・MSP 問題の自然な拡張である．数の矩形領域 (リストのリスト) が与えられた時，MSS2 はその中の部分矩形領域の要素の和の最大値を求め，MSP2 は積の最大値を求める．MSS2 問題に関しては，[22] に 11 個の関数の組の導入による並列アルゴリズムの導出が述べられているが，本論文で対象とするような逐次アルゴリズムの導出に関しては，MSP2 問題を含め，公表されたものはない．

これらの問題を解く素朴で明白な初期プログラム $mss2$・$msp2$ は，一次元の場合と同様に，すべての部分矩形領域を列挙してリストとする関数 $segs2$，矩形領域の要素の和を求める関数 $sum2$，積を求める関数 $prod2$ を使って次のように記述できる．

$$
\begin{aligned}
&mss2 = max \circ map\, sum2 \circ segs2\\
&msp2 = max \circ map\, prod2 \circ segs2\\
&segs2\,[\,] = [[\,]]\\
&segs2\,(xs : xss)\\
&\quad = concat\,(tops\,(xs : xss)) +\!\!+ segs2\,xss\\
&tops\,[\,] = [[\,], \ldots, [\,]]\\
&tops\,(xs : xss)\\
&\quad = zipwith\,(\lambda(u, v)\,.\,[u] : map\,(u :)\,v)\\
&\qquad\qquad (segs\,xs)\,(tops\,xss)
\end{aligned}
$$

ただしここでは $mss2$，$msp2$ の引数には，等しい長さの数のリストからなるリスト (矩形領域として整合性のとれたデータ) が与えられるものと仮定している．また $zipwith$ は，二つのリストの対応する順番の要素に対して，与えられた関数を適用した結果をリストとして返すような標準関数である．

$segs2$ は，下請けとして $tops$ を呼ぶ．$tops$ は，引数として与えられた矩形領域の部分矩形領域のうち，上辺が引数の上辺と接するものだけを列挙してリストとして返す．そのため，$tops$ は $segs$ をさらに下請けとして呼ぶ．ここでは定義がいたずらに複雑になるのを避けるため，$tops\,[\,]$ の返す値は，対象とする矩形領域 (xss とする) の上辺 ($head\,xss$) に対して $segs$ が返すリスト ($segs\,(head\,xss)$) と同じ長さの $[\,]$ だけからなるリストとした．これを上の定義では $[[\,], \ldots, [\,]]$ で表記している．以下で示す運算過程においても，リスト中の $\cdots$ はこの長さのリストを表現するものと約束する．

本論文の手法をこれらの問題に適用することにより，MSS2 問題は漸次的組化をすることなく，また MSP2 問題の場合は漸次的組化を行って，初期プログラムからの系統的運算によって，効率のよいプログラムを導出することができる．MSP2 問題で漸次的組化が必要なのは，MSP 問題の場合と同様の難しさ (演算子の非単調性) を内在しているためである．計算量に関しては，矩形領域の一辺の大きさを N とすると，いずれの場合も，初期プログラムが $O(N^6)$ 必要だったのに対し，導出したプログラムは $O(N^3)$ へと軽減される．

5.1 二次元最大部分列和問題

5.1.1 $segs2$ と $tops$ の組化

$segs2$ は単独には Cata にならず，$tops$ と同一のデータを辿るので，これらを組化変換する．

$$
\begin{aligned}
&segs2\,(xs : xss) = \psi_{segs2}\,(xs,\ (segs2\,xss,\ tops\,xss))\\
&tops\,(xs : xss) = \psi_{tops}\,(xs,\ (segs2\,xss,\ tops\,xss))
\end{aligned}
$$

という対応を考えると，ψ_{segs2}，ψ_{tops} は

$$
\begin{aligned}
&\psi_{segs2}\,(xs,\ (yss, zss))\\
&\quad = concat\,(zipwith\,f\,(segs\,xs)\,zss) +\!\!+ yss\\
&\psi_{tops}\,(xs,\ (yss, zss)) = zipwith\,f\,(segs\,xs)\,zss\\
&f\,(u, v) = [u] : map\,(u :)\,v
\end{aligned}
$$

と定めることができる．これらを用いて，$segs2 \triangle tops$ は

$$
F_{mss2} = (\!|\,([[\,]],\ [[\,], \ldots, [\,]]),\ \psi_{segs2} \triangle \psi_{tops}\,|\!)
$$

という Cata で表現される．

5.1.2 $map\,sum2$ の融合

h_2 を適当な関数として，$(map\,sum2 \times h_2) \circ F_{mss2}$ を融合する．まず，定理の前提条件

$$
\begin{aligned}
&map\,sum2\,(\psi_{segs2}\,(xs,\ (yss, zss)))\\
&\quad = \psi'_{segs2}\,(xs,\ (map\,sum2\,yss,\ h_2\ zss))
\end{aligned}
$$

に関する運算を次のように進める．

$$
\begin{aligned}
&map\,sum2\,(\psi_{segs2}\,(xs,\ (yss, zss)))\\
&= \{\ \psi_{segs2}\text{の定義}\ \}\\
&\quad map\,sum2\,(concat\,(zipwith\,f\,(segs\,xs)\,zss)\\
&\qquad\qquad +\!\!+ yss)\\
&= \{\ map\text{-}append\text{則}\ \}\\
&\quad map\,sum2\,(concat\,(zipwith\,f\,(segs\,xs)\,zss))\\
&\quad +\!\!+ map\,sum2\,yss\\
&= \{\ map\text{-}concat\text{則}\ \}
\end{aligned}
$$

$$
\begin{aligned}
&concat\\
&\quad (map\,(map\,sum2)\,(zipwith\,f\,(segs\,xs)\,zss))\\
&\quad +\!\!+\ map\,sum2\,yss\\
=\ &\{\ map\text{–}zipwith\text{則}\ \}\\
&concat\,(zipwith\,(map\,sum2\circ f)\,(segs\,xs)\,zss)\\
&+\!\!+\ map\,sum2\,yss
\end{aligned}
$$

ここで，上式の $zipwith$ の第一引数は，

$$
\begin{aligned}
&map\,sum2\circ f\\
=\ &\{\ f\text{の定義}\ \}\\
&map\,sum2\circ(\lambda(u,v)\ .\ [u] : map\,(u:)\,v)\\
=\ &\{\ map\text{の定義}\ \}\\
&\lambda(u,v)\ .\ sum2\,[u] : map\,sum2\,(map\,(u:)\,v)\\
=\ &\{\ map\text{分配則}\ \}\\
&\lambda(u,v)\ .\ sum2\,[u] : map\,(sum2\circ map\,(u:))\,v\\
=\ &\{\ sum2\text{の定義}\ \}\\
&\lambda(u,v)\ .\ sum\,u : map\,((sum\,u+)\circ sum2)\,v\\
=\ &\{\ map\text{分配則}\ \}\\
&\lambda(u,v)\ .\ sum\,u : map\,(sum\,u+)\,(map\,sum2\,v)
\end{aligned}
$$

と変形されるので，

$$
\begin{aligned}
&map\,sum2\,(\psi_{segs2}\,(xs,\ (yss,zss)))\\
=\ &concat\,(zipwith\,(\lambda(u,v)\ .\\
&\qquad sum\,u : map\,(sum\,u+)\,(map\,sum2\,v))\\
&(segs\,xs)\,zss)\ +\!\!+\ map\,sum2\,yss\\
=\ &\{\ zipwith\text{–}map\text{則},\ g(u,v)=u:map(u+)\,v\ \}\\
&concat\,(zipwith\,g\,(map\,sum\,(segs\,xs))\\
&\qquad\qquad (map\,(map\,sum2)\,zss))\\
&+\!\!+\ map\,sum2\,yss\\
=\ &\{\ \text{定理の前提条件}\ \}\\
&\psi'_{segs2}\,(xs,\ (map\,sum2\,yss,\ h_2\,zss))
\end{aligned}
$$

これより，h_2 と ψ'_{segs2} は次のように定めればよい．

$$
\begin{aligned}
&h_2 = map\,(map\,sum2)\\
&\psi'_{segs2}\,(xs,\ (ys,zs))\\
&= concat\,(zipwith\,g\,(map\,sum\,(segs\,xs))\,zs) +\!\!+\ ys
\end{aligned}
$$

また，融合定理適用のもうひとつの前提条件

$$
\begin{aligned}
&h_2\,(\psi_{tops}\,(xs,\ (yss,zss)))\\
&\quad = \psi'_{tops}\,(xs,\ (map\,sum2\,yss,\ h_2\,zss))
\end{aligned}
$$

についても上と同様にして，

$$
\begin{aligned}
&h_2\,(\psi_{tops}\,(xs,\ (yss,zss)))\\
=\ &\{\ \text{上と同様}\ \}\\
&zipwith\,g\,(map\,sum\,(segs\,xs))\\
&\qquad (map\,(map\,sum2)\,zss)\\
=\ &\{\ \text{定理の前提条件}\ \}\\
&\psi'_{tops}\,(xs,\ (map\,sum2\,yss,\ h_2\,zss))
\end{aligned}
$$

から，ψ'_{tops} は，

$$
\begin{aligned}
&\psi'_{tops}\,(xs,\ (ys,zs))\\
&= zipwith\,g\,(map\,sum\,(segs\,xs))\,zs
\end{aligned}
$$

と定められる．また，リストが $[\,]$ の場合は，

$$
\begin{aligned}
&map\,sum2\,[[\,]] = [0]\\
&h_2\,[[\,],\ldots,[\,]] = [[\,],\ldots,[\,]]
\end{aligned}
$$

となる．以上より，$(map\,sum2\times h_2)\circ F_{mss2}$ は，次のような Cata へ融合された．

$$
F'_{mss2} = (\!|\,([0],\ [[\,],\ldots,[\,]]),\ \psi'_{segs2}\triangle\psi'_{tops}\,|\!)
$$

5.1.3 max の融合

最後に max との合成を融合するために，h'_2 を適当な関数として $(max\times h'_2)\circ F'_{mss2}$ を融合変換する．

$$
\begin{aligned}
&max\,(\psi'_{segs2}\,(xs,\ (ys,zs)))\\
&\quad = \psi''_{segs2}\,(xs,\ (max\,ys,\ h'_2\,zs))
\end{aligned}
$$

を満たす ψ''_{segs2} と h'_2 を次のように定める．

$$
\begin{aligned}
&max\,(\psi'_{segs2}\,(xs,\ (ys,zs)))\\
=\ &\{\ \psi'_{segs2}\text{の定義}\ \}\\
&max\,(concat\,(zipwith\,g\,(map\,sum\,(segs\,xs))\,zs)\\
&\qquad +\!\!+\ ys)\\
=\ &\{\ max\text{–}append\text{則}\ \}\\
&max(concat\,(zipwith\,g\,(map\,sum\,(segs\,xs))\,zs))\\
&\uparrow max\,ys\\
=\ &\{\ max\text{–}concat\text{則}\ \}\\
&max\,(map\,max\\
&\qquad\qquad (zipwith\,g\,(map\,sum\,(segs\,xs))\,zs))\\
&\uparrow max\,ys\\
=\ &\{\ map\text{–}zipwith\text{則}\ \}\\
&max(zipwith\,(max\circ g)\,(map\,sum\,(segs\,xs))\,zs)\\
&\uparrow max\,ys
\end{aligned}
$$

ここで，$max\circ g$ の部分は次のように変形される．

$$
\begin{aligned}
&max\circ g\\
=\ &\{\ g\text{の定義}\ \}\\
&max\circ(\lambda(u,v)\ .\ u : map\,(u+)\,v)\\
=\ &\{\ max\text{の定義}\ \}\\
&\lambda(u,v)\ .\ u\uparrow max\,(map\,(u+)\,v)\\
=\ &\{\ max\text{–加算則}\ \}\\
&\lambda(u,v)\ .\ u\uparrow(u+max\,v)
\end{aligned}
$$

したがって，

$$\begin{aligned}
&max\,(\psi'_{segs2}\,(xs,\,(ys,\,zs)))\\
&=\ max\,(zipwith\,(\lambda(u,v)\ .\ u\uparrow(u+max\,v))\\
&\qquad\qquad (map\ sum\,(segs\,xs))\ zs)\\
&\quad\uparrow max\,ys\\
&=\ \{\ zipwith\text{-}map\text{則},\ k\,(u,v)=u\uparrow(u+v)\ \}\\
&\quad max\,(zipwith\,k\\
&\qquad\qquad (map\ sum\,(segs\,xs))\,(map\ max\,zs))\\
&\quad\uparrow max\,ys\\
&=\ \{\ 定理の前提條件\ \}\\
&\quad \psi''_{segs2}\,(xs,\,(max\,ys,\,h'_2\ zs))
\end{aligned}$$

となり，ψ''_{segs2} と h'_2 は次のように決定する．

$$\begin{aligned}
&h'_2=map\ max\\
&\psi''_{segs2}\,(xs,\,(y,z))\\
&=\ max\,(zipwith\,k\,(map\ sum\,(segs\,xs))\,z)\uparrow y
\end{aligned}$$

同様にして，他方の前提条件

$$\begin{aligned}
&map\ max\,(\psi'_{tops}\,(xs,\,(ys,zs)))\\
&\quad=\psi''_{tops}\,(xs,\,(max\,ys,\,h'_2\ zs))
\end{aligned}$$

については，

$$\begin{aligned}
&map\ max\,(\psi'_{tops}\,(xs,\,(ys,zs)))\\
&=\ \{\ \psi'_{tops}\ の定義\ \}\\
&\quad map\ max\,(zipwith\,g\,(map\ sum\,(segs\,xs))\ zs)\\
&=\ \{\ 前と同様\ \}\\
&\quad zipwith\,k\,(map\ sum\,(segs\,xs))\,(map\ max\,zs)\\
&=\ \{\ 定理の前提条件\ \}\\
&\quad \psi''_{tops}\,(xs,\,(max\,ys,\,h'_2\ zs))
\end{aligned}$$

より，ψ''_{tops} は

$$\begin{aligned}
&\psi''_{tops}\,(xs,\,(y,z))\\
&=\ zipwith\,k\,(map\ sum\,(segs\,xs))\,z
\end{aligned}$$

と定まった．さらに，

$$\begin{aligned}
&max\,[0]=0\\
&h'_2\,[[],\ldots,[]]=[-\infty,\ldots,-\infty]
\end{aligned}$$

を総合して，$(max\times h'_2)\circ F'_{mss2}$ は次のような Cata に融合された．

$$F''_{mss2}=(\!|\,(0,\,[-\infty,\ldots,-\infty]),\,\psi''_{segs2}\triangle\psi''_{tops}\,|\!)$$

ψ''_{segs2} には ψ''_{tops} に相当する計算が含まれることを考慮し，さらに，$map\ sum\,(segs\,xs)$ の部分に関しては，第3節に示した MSS 問題の運算途中に出現した融合結果 (F'_{mss}) を使い，通常の形の定義に書きかえると，次のような最終結果が得られる．

$$\begin{aligned}
&mss2\,xss=y\ \ \mathsf{where}\ \ (y,z)=F''_{mss2}\,xss\\
&F''_{mss2}\,[\,]=(0,\,[-\infty,\ldots,-\infty])\\
&F''_{mss2}\,(xs:xss)=(max\,p\uparrow y,\,p)\\
&\quad \mathsf{where}\ \ (y,z)=F''_{mss2}\,xss,\ \ (s,t)=F'_{mss}\,xs\\
&\qquad\quad p=zipwith\,(\lambda(u,v)\ .\ u\uparrow(u+v))\,s\,z\\
&F'_{mss}\,[\,]=([0],[0])\\
&F'_{mss}\,(x:xs)=(ps\mathbin{+\!\!+}ys,\,ps)\\
&\quad \mathsf{where}\ \ (ys,zs)=F'_{mss}\,xs\\
&\qquad\quad ps=0:map\,(x+)\,zs
\end{aligned}$$

5.2 二次元最大部分列積問題

二次元最大部分列積問題の初期プログラムは，MSS2 問題での $sum2$ を $prod2$ に置きかえて得られる．

$$msp2=max\circ map\ prod2\circ segs2$$

5.2.1 組化と $map\ prod2$ の融合

$segs2$ と $tops$ の組化 $F_{msp2}=segs2\triangle tops$，$h_2$ を適当な関数として $(map\ prod2\times h_2)\circ F_{msp2}$ の融合までは，MSS2 の場合と，加算 ($segs2$) と乗算 ($prod2$) の違いを除いて，全く同じ運算過程をたどる．$(map\ prod2\times h_2)\circ F_{msp2}$ の融合結果 (F'_{msp2} とする) を次に示す．

$$\begin{aligned}
&F'_{msp2}=(\!|\,([1],\,[[],\ldots,[]]),\,\psi'_{segs2}\triangle\psi'_{tops}\,|\!)\\
&h_2=map\,(map\ prod2)\\
&\psi'_{segs2}\,(xs,\,(ys,zs))\\
&=\ concat\,(zipwith\,g\,(map\ prod\,(segs\,xs))\,zs)\mathbin{+\!\!+}ys\\
&\psi'_{tops}\,(xs,\,(ys,zs))\\
&=\ zipwith\,g\,(map\ prod\,(segs\,xs))\,zs\\
&g\,(u,v)=u:map\,(u*)\,v
\end{aligned}$$

5.2.2 max の融合

次に h'_2 を適当な関数として，$(max\times h'_2)\circ F'_{msp2}$ の融合変換を行う．まず組化 Cata 融合定理の前提条件

$$\begin{aligned}
&max\,(\psi'_{segs2}\,(xs,\,(ys,zs)))\\
&\quad=\psi''_{segs2}\,(xs,\,(max\,ys,\,h'_2\ zs))
\end{aligned}$$

を次のように運算する．

$$\begin{aligned}
&max\,(\psi'_{segs2}\,(xs,\,(ys,zs)))\\
&=\ \{\ \psi'_{segs2}\ の定義\ \}\\
&\quad max\,(concat\,(zipwith\,g\,(map\ prod\,(segs\,xs))\,zs)\\
&\qquad\qquad \mathbin{+\!\!+}ys)\\
&=\ \{\ \mathrm{MSS2}と全く同じ手順\ \}\\
&\quad max\,(zipwith\,(max\circ g)\,(map\ prod\,(segs\,xs))\,zs)
\end{aligned}$$

$\uparrow max\, ys$

ここで，$max \circ g$ の部分は，

$max \circ g$
$= \{\ g$ の定義，max の定義 $\}$
$\lambda(u, v)\ .\ u \uparrow max\,(map\,(u *)\,v)$
$= \{\ max$–乗算則 $\}$
$\lambda(u, v)\ .\ u \uparrow (u * (\mathsf{if}\ u \geq 0\ \mathsf{then}\ max\, v\ \mathsf{else}\ min\, v))$

となる．したがって，

$max\,(\psi'_{segs2}\,(xs,\,(ys, zs)))$
$= max\,(zipwith$
$(\lambda(u, v)\ .\ u \uparrow (u * (\mathsf{if}\ u \geq 0\ \mathsf{then}\ max\, v\ \mathsf{else}\ min\, v)))$
$(map\, prod\,(segs\, xs))\ zs) \uparrow max\, ys$
$= \{\ zipwith3^{\dagger 2}$の導入 $\}$
$max\,(zipwith3$
$(\lambda(u, v, w)\ .\ u \uparrow (u * (\mathsf{if}\ u \geq 0\ \mathsf{then}\ max\, v\ \mathsf{else}\ min\, w)))$
$(map\, prod\,(segs\, xs))\ zs\ zs) \uparrow max\, ys$
$= \{\ zipwith3$–map則，
$k_1\ (u, v, w)$
$= u \uparrow (u * (\mathsf{if}\ u \geq 0\ \mathsf{then}\ v\ \mathsf{else}\ w))\ \}$
$max\,(zipwith3\, k_1$
$(map\, prod\,(segs\, xs))$
$(map\, max\, zs)\,(map\, min\, zs)) \uparrow max\, ys$

ここで，zs に対する関数適用 $(map\, max\, zs$ および $map\, min\, zs)$ が二箇所に分かれて出現するので，これらを（漸次的）組化することにより，

上式
$= max\,(zipwith3\, k_1\,(map\, prod\,(segs\, xs))\ s\ t)$
$\uparrow max\, ys$
$\mathsf{where}\ (s, t) = (map\, max \triangle map\, min)\ zs$

となる．これが $\psi''_{segs2}\,(xs,\,(max\, ys,\, h'_2\, zs))$ と等しくなるように h'_2 と ψ''_{segs2} を次のように決定する．

$h'_2 = map\, max \triangle map\, min$
$\psi''_{segs2}\,(xs,\,(y,\,(s, t)))$
$= max\,(zipwith3\, k_1\,(map\, prod\,(segs\, xs))\ s\ t) \uparrow y$

†2 $zipwith$ を 3 個のリストへ拡張したもので，変換規則についても $zipwith$–map 則と同様のもの（“$zipwith3$–map 則”と呼ぶ）が成立する．

次に，もう一方の前提条件

$h'_2\,(\psi'_{tops}\,(xs,\,(ys, zs)))$
$= \{\ h'_2$ の定義 $\}$
$(map\, max\,(\psi'_{tops}\,(xs,\,(ys, zs))),$
$map\, min\,(\psi'_{tops}\,(xs,\,(ys, zs))))$
$= \psi''_{tops}\,(xs,\,(max\, ys,\, h'_2\, zs))$

が満足されるように，ψ''_{tops} を決定する．この式の値の組の第一要素の運算は，次のようになる．

第一要素
$= map\, max\,(\psi'_{tops}\,(xs,\,(ys, zs)))$
$= \{\ \psi'_{tops}$の定義 $\}$
$map\, max\,(zipwith\, g\,(map\, prod\,(segs\, xs))\ zs)$
$= \{$ 前と同様 $\}$
$zipwith3\, k_1\,(map\, prod\,(segs\, xs))\ s\ t$
$\mathsf{where}\ (s, t) = (map\, max \triangle map\, min)\ zs$

次に組の第二要素については，第一要素と max と min が入れかわった形となる．

$map\, min\,(\psi'_{tops}\,(xs,\,(ys, zs)))$
$= \{$ 上と同様
$k_2\ (u, v, w)$
$= u \downarrow (u * (\mathsf{if}\ u \geq 0\ \mathsf{then}\ w\ \mathsf{else}\ v))\ \}$
$= zipwith3\, k_2\,(map\, prod\,(segs\, xs))\ s\ t$
$\mathsf{where}\ (s, t) = (map\, max \triangle map\, min)\ zs$

以上より，

$\psi''_{tops}\,(xs,\,(y,\,(s, t)))$
$= (zipwith3\, k_1\,(map\, prod\,(segs\, xs))\ s\ t,$
$zipwith3\, k_2\,(map\, prod\,(segs\, xs))\ s\ t)$

と定める．さらに，[] の場合は

$max\,[1] = 1$
$h'_2\,[[\,], \ldots, [\,]] = ([-\infty, \ldots, -\infty], [\infty, \ldots, \infty])$

なので，$(max \times h'_2) \circ F'_{msp2}$ は

$F''_{msp2} = (\!|\,(1,\,([-\infty, \ldots, -\infty], [\infty, \ldots, \infty])),$
$\psi''_{segs2} \triangle \psi''_{tops}\,|\!)$

という Cata に融合される．ψ''_{segs2} と ψ''_{tops} には同じ計算が含まれることを考慮し，$map\, prod\,(segs\, xs)$ については，第 4 節の MSP 問題の運算の途中経過で得られた融合結果 (F'_{msp}) を使い，通常の形の定義に書きかえると，次のような最終結果が得られる．

$msp2\, xss = y\ \mathsf{where}\ (y, z) = F''_{msp2}\, xss$
$F''_{msp2}\,[\,] = (1,\,([-\infty, \ldots, -\infty], [\infty, \ldots, \infty]))$

$$
\begin{aligned}
&F''_{msp2}\,(xs : xss) = (max\,p \uparrow y,\ (p, q))\\
&\quad \text{where}\\
&\qquad (y, (s, t)) = F''_{msp2}\,xss,\ \ (m, n) = F'_{msp}\,xs\\
&\qquad p = zipwith3\,k_1\ m\ s\ t\\
&\qquad q = zipwith3\,k_2\ m\ s\ t\\
&\qquad k_1\,(u, v, w)\\
&\qquad\quad = u \uparrow (u * (\text{if } u \geq 0 \text{ then } v \text{ else } w))\\
&\qquad k_2\,(u, v, w)\\
&\qquad\quad = u \downarrow (u * (\text{if } u \geq 0 \text{ then } w \text{ else } v))\\
&F'_{msp}\,[\,] = ([1], [1])\\
&F'_{msp}\,(x : xs) = (ps +\!\!+ ys,\ ps)\\
&\quad \text{where } (ys, zs) = F'_{msp}\,xs\\
&\qquad\quad ps = 1 : map\,(x *)\,zs
\end{aligned}
$$

以上のようにして，本論文で提案する手法を用いた運算により，効率のよいアルゴリズムが導出された．

6 関連研究

Bird [3]も，BMF の手法に基づく“運算”の例題として，MSS・MSP 両問題をとりあげて，第 3 節で示した最終結果と実質的に同等な線形時間の効率のよいアルゴリズムの導出に成功している．MSS 問題の導出過程の概略は，次の通り．

$$
\begin{aligned}
&mss\\
&= max \circ map\,sum \circ segs\\
&= \{\ \text{途中省略}\ \}\\
&\quad max \circ map\,(max \circ map\,sum \circ tails) \circ inits\\
&= \{\ x \ominus y = (x + y) \uparrow 0,\ \text{ホーナー則(下参照)}\ \}\\
&\quad max \circ map\,(foldl\,(\ominus)\,0) \circ inits\\
&= \{\ \text{途中省略},\\
&\qquad (u, v) \oslash x = ((v + x) \uparrow 0,\ u \uparrow (v + x) \uparrow 0)\ \}\\
&\quad fst \circ foldl\,(\oslash)\,(0, 0)
\end{aligned}
$$

ここで，$tails$ はリストの $tail$ を順次とってリストとして返す標準関数，fst は 組の第一要素を返す関数である．この導出における急所は，多項式値の計算方法として有名なホーナーの方法の一般化である“ホーナー則”

$$
\begin{aligned}
&(x \oplus y) \otimes z = (x \otimes z) \oplus (y \otimes z),\ e \oplus x = x \quad \Longrightarrow\\
&foldl\,(\oplus)\,e \circ map\,(foldl\,(\otimes)\,d) \circ tails = foldl\,(\ominus)\,d\\
&\text{where } x \ominus y = (x \otimes y) \oplus d
\end{aligned}
$$

を適用する場面である．前提部をみてわかる通り，この規則を適用するためには，演算子 $\oplus$ と $\otimes$ が分配則を満たさなければならない．この問題の場合，$max = foldl\,(\uparrow)\,(-\infty)$，$sum = foldl\,(+)\,0$ なので，$\uparrow$ と $+$ が分配則

$$(x \uparrow y) + z = (x + z) \uparrow (y + z)$$

を自然に満たし，ホーナー則を適用することができる．ところが，MSP 問題に対して同様の変換を適用しようとすると，途中までは変換を進めることができるが，$*$ と $\uparrow$ がこのような分配則を満たしていないため，ホーナー則を適用しようとする段で行き詰まる．(この事実は，第 4 節で指摘した乗算の非単調性に対応する．)

MSP 問題の変換をさらに進めるには，ホーナー則の前提条件を満足する演算子を“発明”するために，$max \circ map\,prod \circ tails$ と $min \circ map\,prod \circ tails$ を組にした関数を考える必要がある．この二つを組にすることに思い至るための明確な手掛りはなく，変換を進める人の深い洞察力を必要とする．

BMF がこのような演算子の“職人技”的な発明，ホーナー則の適用といった障壁の高いステップを必要とするのに対し，本論文の方法では，Cata の形にした後の融合定理の適用，必要に応じた漸次的組化という指針に従うことにより，特別な関数・演算子を発明することなく，より系統的に運算を進めることができる．

融合と組化変換に関する研究 [5] [6]の多くは，コンパイラによる最適化を念頭においており，両変換手法は独立に論じられることが多かった．これに対し本論文の手法では，人間の手による運算ではあるが，融合と組化をうまく組み合わせ，相乗的な効果を得ることに成功した．

我々の論文 [13]は，組化変換に関して“組化可能”な関数のクラスを定義し，このクラスに属す関数に対する組化アルゴリズムを提示した．この組化アルゴリズムは，本論文の手法における組化変換を行う場面で適用することができる．このことは，本論文の手法を自動化へと発展させていく際の手掛りを与えている．

7 おわりに

本論文では，プログラム発展という立場から，融合と組化という独立したプログラム変換手法を組み合わせて，構成的アルゴリズム論の考えに基づく運算を進める方法を提示した．提案した手法では，対象とする関数を

組化して Cata の形を導出してから組化 Cata 融合定理を繰返し適用する．融合定理適用の際，必要に応じて漸次的組化を行うことによって，素朴な初期プログラムから系統的な運算を進め，効率のよいプログラムへ変換できることを，自明でない具体例を通して示した．

構成的アルゴリズム論に基づく運算による効率のよい既知のアルゴリズムの導出，あるいは，新しいアルゴリズムの発見については，本論文であげた例の他に，たとえばデータマイニングにおける頻出集合問題 [14]などにおいても成果を示してきている．今後，本研究のさらなる進展により，様々なアルゴリズムの導出が可能となり，効率のよいアルゴリズムへの発展機構解明の手掛かりが得られることが期待される．さらに，本論文の手法をはじめとするプログラム運算の自動化に関しても研究を進めていく必要がある．

謝辞

本研究は，文部省科学研究費補助金特定領域研究 (A)(1)「発展機構を備えたソフトウェアの構成原理の研究」の援助を受けて行われた．

参考文献

[1] Bentley, M. J.: Programming Pearls, *Comm. ACM*, Vol. 27, No. 9 (1984), pp. 865–871.

[2] Bird, R.: An Introduction to the Theory of Lists, *Logic of Programming and Calculi of Discrete Design*, NATO ASI Series 36, Springer-Verlag, 1987, pp. 5–42.

[3] Bird, R.: Lecture Notes on Theory of Lists, STOP Summer School on Constructive Algorithmics, 1987.

[4] Bird, R.: *Introduction to Functional Programming using Haskell*, Prentice-Hall, 1998.

[5] Chin, W. N.: Safe Fusion of Functional Expressions, *Proc. 7-th ACM Lisp and Functional Programming*, ACM Press, 1992, pp. 11–20.

[6] Chin, W. N.: Towards an Automated Tupling Strategy, *Proc. PEPM '93*, ACM Press, 1993, pp. 119–132.

[7] Fokkinga, M.: Tupling and Mutumorphisms, *Squiggolist*, Vol. 1, No. 4 (1989).

[8] Fokkinga, M.: *Law and Order in Algorithmics*, Ph.D. thesis, Dept. Inf. University of Twente, 1992.

[9] Fukuda, T., Morimoto, Y., Morishita, S. and Tokuyama, T.: Data Mining Using Two Dimensional Optimized Association Rules: Scheme, Algorithms, and Visualization, *Proc. SIGMOD '96*, ACM Press, 1996, pp. 13–23.

[10] Gill, A., Launchbury, J. and Jones, S. P.: A Short Cut to Deforestation, *Proc. FPCA '93*, ACM Press, 1993, pp. 223–232.

[11] Hagino, T.: A Typed Lambda Calculus with Categorical Type Constructors, *Category Theory and Computer Science*, Lecture Notes in Computer Science 283, Springer-Verlag, 1988, pp. 140–157.

[12] Hu, Z., Iwasaki, H. and Takeichi, M.: Deriving Structural Hylomorphisms from Recursive Definitions, *Proc. ICFP '96*, ACM Press, 1996, pp. 73–82.

[13] Hu, Z., Iwasaki, H., Takeichi, M. and Takano, A.: Tupling Calculation Eliminates Multiple Data Traversals, *Proc. ICFP '97*, ACM Press, 1997, pp. 164–175.

[14] Hu, Z., Chin W. N. and Takeichi, M.: Calculating a New Data Mining Algorithm for Market Basket Analysis, *Proc. PADL2000*, Lecture Notes in Computer Science 1753, Springer-Verlag, 1999, pp. 169–184.

[15] 岩崎英哉：構成的アルゴリズム論，コンピュータソフトウェア，Vol. 15, No. 6 (1998), pp. 57–70.

[16] Launchbury, J. and Sheard, T.: Warm Fusion: Deriving Build-Catas from Recursive Definitions, *Proc. FPCA '95*, ACM Press, 1995, pp. 314–323.

[17] Malcolm, G: Data structures and Program Transformation, *Science of Computer Programming*, Vol. 14, Nos. 2–3 (1990), pp. 255–279.

[18] Meijer, E., Fokkinga, M. and Paterson, R.: Functional Programming with Bananas, Lenses, Envelopes and Barbed Wire, *Proc. FPCA '91*, Lecture Notes in Computer Science 523, Springer-Verlag, 1991, pp. 124–144.

[19] Onoue, Y., Hu, Z., Iwasaki, H. and Takeichi, M.: A Calculational Fusion System HYLO, *Proc. IFIP TC2 Working Conference on Algorithmic Languages and Calculi*, Chapman & Hall, 1997, pp. 76–106.

[20] 尾上能之，胡振江，岩崎英哉，武市正人：プログラム融合変換の実用的有効性の検証，コンピュータソフトウェア，Vol. 17, No. 3 (2000), pp. 81–85.

[21] Sheard, T. and Fegaras, L.: A Fold for All Seasons, *Proc. FPCA '93*, ACM Press, 1993, pp. 233–242.

[22] Smith, D. R.: Applications of a Strategy for Designing Divide-and-conquer Algorithms, *Sci. Comput. Program.*, Vol. 8, No. 3 (1987), pp. 213–229.

[23] Takano, A. and Meijer, E.: Shortcut Deforestation in Calculational Form, *Proc. FPCA '95*, ACM Press, 1995, pp. 306–313.

[24] Wadler, P.: Deforestation: Transforming Programs to Eliminate Trees, *Proc. ESOP*, Lecture Notes in Computer Science 300, Springer-Verlag, 1988, pp. 344–358.