D6-3

変換部品の組合せによるプログラムの最適化

Deriving Efficiemt Programs by Fusion and Tupling

岩崎 英哉 Hideya IWASAKI　胡 振江 Zhenjiang HU　武市 正人 Masato TAKEICHI

東京大学大学院工学系研究科
Graduate School of Engineering, University of Tokyo

概 要

本稿では，複数の独立したプログラム変換手法，具体的には融合 (Fusion) と組化 (Tupling) を順番に適用することにより，プログラムを効率のよいものへ系統的に変換可能であることを示す．融合は計算途中で生成される中間的なデータ構造を削除し，組化は同一のデータ構造を重複して辿ることを軽減する．これらを組み合せれば，素朴な初期プログラムから，時間空間の両面において効率のよいプログラムを導出できる．ここでは，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，関数合成間で中間的なデータが生成・消費されること，同一のデータ構造を複数の関数が辿ることに起因する非効率性を持つ．

この問題を解決するプログラム変換手法に，融合 (Fusion) [3] と組化 (Tupling) [4] [5] がある．融合は関数合成をひとつの関数にまとめることにより，前者の問題を解決し，組化は同一の実引数を持つ関数を組 (Tuple) にして，後者の問題を解決する．

この種の変換を進める方法論に，BMF(Bird-Meertens Formalism) [2] がある．BMF では，変換対象の関数を *map*・*foldr* などの基本関数の組合せで表現し，少数の強力な変換定理を用いてプログラムを構成的に変換する．これは有効な方法であるが，対象関数のどの部分にどの定理をいつ適用するかは，難しい問題である．さらに，変換定理を単純には適用できない場面では，補助関数を定義したり組を作るなどの発見的手法が必要である．この点は変換を進める人の深い洞察力に依存しているため，BMF では系統的な変換を行うのが困難である．

本稿では，融合と組化を独立した "変換部品" と捉え，はじめに融合を行った後に，組化を進めることにより，より系統的なプログラム変換が可能であることを示す．この手法を "FT 変換" と呼ぶ．はじめに最大部分列和問題を具体例として，本手法を説明する．その後，最大部分列積問題，さらに二次元最大部分列和 / 積問題にも本手法を適用し，効率のよいプログラムの導出に有用であることを示す．

2 一 / 二次元最大部分列和 / 積問題

2.1 一次元の場合

最大部分列和 (Maximum Segment Sum，略して 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．

この問題を解くための初期プログラム $mss\ xs$ は，xs のすべての部分列を列挙してリストを作る関数 *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．

$$mss\ xs = max\,(map\ sum\,(segs\ xs))$$

最大部分列積 (Maximum Segment Product，略して MSP) 問題は，MSS での "和" を "積" にした (空部分列の積は 1 とする) ものである．リストの要素の積を求める関数を $prod$ とすれば，MSP の初期プログラム msp は mss と同様にして得られる．

$$msp\ xs = max\,(map\ prod\,(segs\ xs))$$

両者とも，初期プログラムの計算量はリストの長さを n として $O(n^3)$ であり，仕様の上では加算と乗算が異なるだけの大変良く似た問題である．しかし後述するように，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．

2.2 二次元の場合

二次元最大部分列和 (MSS2) 問題・部分列積 (MSP2) 問題は，MSS・MSP 問題の自然な拡張である．整数の矩形領域 (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．

これらの問題を解く初期プログラム $mss2$・$msp2$ 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 $prod2$ を使って記述できる (図 5 参照)．MSP2 問題のプログラム変換にも，MSP の場合と同様の理由による困難さが内在する．

3 BMF によるアプローチ

BMF では，対象とする関数を基本的な関数*の組合せで表現し，これらの基本関数に関する変換規則を順次適用してプログラムを変換する．図 1 に MSS・MSP 問題を記述する関数の定義を，図 2 に代表的な変換規則をいくつか†示す．規則 (R2–4) は多項式値を求める標準的な方法として知られている "ホーナー則" の一般化である．

BMF による MSS 問題の変換は次の通り．最終的なプログラムが必要な計算量は，$O(n)$ である．

* ここで用いる基本関数は，特に断わらない限り，関数型言語 Miranda の標準関数 [1]と同じ仕様とする．

† $\circ$ は関数合成を，$\uparrow$ と $\downarrow$ は大きい値と小さい値を返す二項演算子を，$\oplus$，$\ominus$ などは二項演算子を，fst は組の第一要素を返す関数を表す．また本稿では，二項演算子を括弧で囲むセクション化の記法を使う．

(D1–1) $mss = max \circ map\ sum \circ segs$

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

(D1–2) $segs = concat \circ map\ inits \circ tails$

(D1–3) $sum = foldr\,(+)\,0,\quad prod = foldr\,(*)\,1$

(D1–4) $max = foldr\,(\uparrow)\,(-\infty),\quad min = foldr\,(\downarrow)\,\infty$

(D1–5) $concat = foldr\,(+\!\!+)\,[\,]$

ここで $inits\,[x_0, x_1, \cdots, x_{n-1}]$

$= [[\,], [x_0], [x_0, x_1], \cdots, [x_0, x_1, \cdots, x_{n-1}]]$

$tails\,[x_0, x_1, \cdots, x_{n-1}]$

$= [[x_0, x_1, \cdots, x_{n-1}], [x_1, \cdots, x_{n-1}], \cdots, [x_{n-1}], [\,]]$

図 1 BMF における MSS・MSP 問題の定義

(R2–1) $map\,f \circ concat = concat \circ map\,(map\,f)$

(R2–2) $max \circ concat = max \circ map\ max$

(R2–3) $map\,f \circ map\,g = map\,(f \circ g)$

(R2–4) $z \otimes (x \oplus y) = (z \otimes x) \oplus (z \otimes y),\ x \oplus e = x \Longrightarrow$

$foldr\,(\oplus)\,e \circ map\,(foldr\,(\otimes)\,d) \circ inits = foldr\,(\ominus)\,d$

where $x \ominus y = d \oplus (x \otimes y)$

(R2–5) $map\,(foldr\,(\oplus)\,e) \circ tails = scanr\,(\oplus)\,e$

(R2–6) $foldr\,(\otimes)\,e \circ scanr\,(\oplus)\,d = fst \circ foldr\,(\oslash)\,(d \otimes e,\ d)$

where $x \oslash (u, v) = (w \otimes x, w),\ w = x \oplus v$

図 2 BMF で用いる変換規則

$$\begin{aligned}
mss &= max \circ map\ sum \circ \\
&\qquad concat \circ map\ inits \circ tails && \text{(D1–1,2)}\\
&= max \circ concat \circ \\
&\qquad map\,(map\ sum) \circ map\ inits \circ tails && \text{(R2–1)}\\
&= max \circ map\ max \circ \\
&\qquad map\,(map\ sum) \circ map\ inits \circ tails && \text{(R2–2)}\\
&= max \circ map\,(max \circ map\ sum \circ inits) \circ \\
&\qquad tails && \text{(R2–3)}\\
&= max \circ map\,(foldr\,(\ominus)\,0) \circ tails && \text{(R2–4)}\\
&\qquad \text{where } x \ominus y = 0 \uparrow (x + y) \\
&= fst \circ foldr\,(\oslash)\,(0 \uparrow -\infty, 0) && \text{(R2–5,6)}\\
&\qquad \text{where } x \oslash (u, v) = (w \uparrow u,\ w),\ w = 0 \uparrow (x + v) \\
&= fst \circ foldr\,(\oslash)\,(0, 0)
\end{aligned}$$

MSP 問題も，途中までは上と同様の変換を進めることができるが，ホーナー則 (R2-4) を適用する段で行き詰まる．なぜなら，MSS 問題の時には，$+$ と $\uparrow$ がホーナー則の前提条件である分配則

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

を満たすが，MSP 問題の時には，$*$ と $\uparrow$ がこの前提条件を満たしていないためである．

MSP 問題の変換を更に進めるには，$max \circ map\ prod \circ inits$ と $min \circ map\ prod \circ inits$ を組にした関数を考え，ホーナー則の前提条件を満足する演算子を定義する必要がある．このように，BMF

(R3–1) $map\ f\ (xs +\!\!+\ ys) = map\ f\ xs +\!\!+\ map\ f\ ys$
(R3–2) $max\ (xs +\!\!+\ ys) = max\ xs \uparrow max\ ys$
(R3–3) $map\ f\ (map\ g\ xs) = map\ (f \circ g)\ xs$
(R3–4) $max\ (map\ (x+)\ xs) = x + max\ xs$
(R3–5) $max\ (map\ (x*)\ xs) = x \geq 0\ \rightarrow x * max\ xs,\ x * min\ xs$
(R3–6) $min\ (map\ (x*)\ xs) = x \geq 0\ \rightarrow x * min\ xs,\ x * max\ xs$
(R3–7) $map\ f\ (zip\ g\ xs\ ys) = zip\ (f \circ g)\ xs\ ys$
(R3–8) $zip(\lambda(x,y)\ .\ f\ (g\ x,\ h\ y))\ xs\ ys = zip\ f\ (map\ g\ xs)\ (map\ h\ ys)$

図3　FT 変換で用いる変換規則

(D4–1) $mss\ xs = max\ (map\ sum\ (segs\ xs))$
$msp\ xs = max\ (map\ prod\ (segs\ xs))$
(D4–2) $segs\ [\,] = [[\,]],\quad segs\ (x : xs) = inits\ (x : xs) +\!\!+\ segs\ xs$
(D4–3) $sum\ [\,] = 0,\quad sum\ (x : xs) = x + sum\ xs$
$prod\ [\,] = 1,\quad prod\ (x : xs) = x * prod\ xs$
(D4–4) $max\ [\,] = -\infty,\quad max\ (x : xs) = x \uparrow max\ xs$
$min\ [\,] = \infty,\quad min\ (x : xs) = x \downarrow min\ xs$
(D4–5) $inits\ [\,] = [[\,]],\quad inits\ (x : xs) = [\,] : map\ (x :)\ (inits\ xs)$
(D4–6) $map\ f\ [\,] = [\,],\quad map\ f\ (x : xs) = f\ x : map\ f\ xs$

図4　FT 変換における MSS・MSP 問題の定義

による変換には深い洞察力を必要とし，簡単に定式化・系統化することはできない．

4 FT 変換

本稿で提案する FT 変換では，融合と組化の二種類の変換を，二段階に分けて適用する．はじめに，次の変換を変換対象関数に対して施す．その結果，中間データ構造を生成しないように融合される．

- 関数定義の左辺から右辺への展開 (unfold)
- 関数合成に関する変換規則 (図 3 ‡) の適用
- 必要に応じて補助関数の定義，定義の右辺から左辺への畳み込み (fold)

漸次的に導入した補助関数に対しても，同様の融合を行う．融合が終了すると，変換対象関数と補助関数による相互再帰定義が得られるので，第二段階としてこれらを組にして，重複した関数呼出しをまとめる．ここでは，MSS・MSP 問題 (図 4) を例題として，その手法を具体的に説明する．

4.1 MSS 問題

第一段階 (融合)

$mss\ [\,] = max\ (map\ sum\ [[\,]])$ (D4–1,2)
$= 0$ (D4–6,3,4)

$mss\ (x : xs)$
$= max\ (map\ sum\ (inits\ (x : xs) +\!\!+\ segs\ xs))$ (D4–1,2)
$= max\ (map\ sum\ (inits\ (x : xs)) +\!\!+$
$map\ sum\ (segs\ xs))$ (R3–1)
$= max\ (map\ sum\ (inits\ (x : xs))) \uparrow$
$max\ (map\ sum\ (segs\ xs))$ (R3–2)

ここで，新しい補助関数

$mxsi\ xs = max\ (map\ sum\ (inits\ xs))$

を導入し，$mxsi$ について上と同様の変換を行う．

$mxsi\ [\,] = 0$ (D4–5,6,3,4)
$mxsi\ (x : xs)$
$= max\ (sum\ [\,] :$
$map\ sum\ (map\ (x :)\ (inits\ xs)))$ (D4–5,6)
$= 0 \uparrow max\ (map\ (sum \circ (x :))\ (inits\ xs))$ (D4–4,R3–3)
$= 0 \uparrow max\ (map\ ((x +) \circ sum)\ (inits\ xs))$ (D4–3)
$= 0 \uparrow (x + max\ (map\ sum\ (inits\ xs)))$ (R3–4)
$= 0 \uparrow (x + mxsi\ xs)$

上の $mxsi$ の結果を mss に反映させると，次のような相互再帰の定義が得られる．

$mss\ [\,] = 0,\ mss\ (x : xs) = (0 \uparrow (x + mxsi\ xs)) \uparrow mss\ xs$
$mxsi\ [\,] = 0,\ mxsi\ (x : xs) = 0 \uparrow (x + mxsi\ xs)$

第二段階 (組化)

mss と $mxsi$ の組 $msstup\ xs = (mss\ xs,\ mxsi\ xs)$ を考え，重複した関数呼出しを除去する．

$mss\ xs = fst\ (msstup\ xs)$
$msstup\ [\,] = (mss\ [\,],\ mxsi\ [\,])$
$= (0, 0)$
$msstup\ (x : xs)$
$= (mss\ (x : xs),\ mxsi\ (x : xs))$
$= ((0 \uparrow (x + mxsi\ xs)) \uparrow mss\ xs,\ 0 \uparrow (x + mxsi\ xs))$
$= (w \uparrow u,\ w)$
where $(u, v) = msstup\ xs,\ w = 0 \uparrow (x + v)$

以上で $O(n)$ のプログラムが得られた．これは，BMF で得られた最終プログラムと等価である．

4.2 MSP 問題

第一段階 (融合)

$msp\ [\,] = 1$ (D4–1,2,6,3,4)
$msp\ (x : xs) = max\ (map\ prod\ (inits\ (x : xs))) \uparrow$
$max\ (map\ prod\ (segs\ xs))$ (D4–1,2,R3–1,2)

ここで，補助関数

$mxpi\ xs = max\ (map\ prod\ (inits\ xs))$

を新たに導入し，上と同様の変換を行う．

$mxpi\ [\,] = 1$ (D4–6,3,4)
$mxpi\ (x : xs) = 1 \uparrow max\ ($
$map\ ((x *) \circ prod)\ (inits\ xs))$ (D4–5,6,4,3,R3–3)

‡ $e_1 \rightarrow e_2,\ e_3$ は if–then–else を表す．

(D5–1) $mss2\,xss = max\,(map\,sum2\,(segs2\,xss))$

$msp2\,xss = max\,(map\,prod2\,(segs2\,xss))$

(D5–2) $segs2\,[\,] = [[\,]]$

$segs2\,(xs : xss) = concat\,(tops\,(xs : xss)) +\!\!+ segs2\,xs$

(D5–3) $tops\,[\,] = [[\,], \cdots, [\,]]$

$tops\,(xs : xss) = zip\,(\lambda(x, y)\,.\,[x] : map\,(x :)\,y)$
$(segs\,xs)\,(tops\,xss)$

(D5–4) $sum2\,[\,] = 0,\ sum2\,(xs : xss) = sum\,xs + sum2\,xss$

$prod2\,[\,] = 1,\ prod2\,(xs : xss) = prod\,xs * prod2\,xss$

$[e, \cdots, e]$ の長さは，対象の矩形を r として $segs\,(hd\,r)$ と同じとする．

図 5　FT 変換における MSS2・MSP2 問題の定義

$$= 1 \uparrow (x \geq 0 \rightarrow x * max\,(map\,prod\,(inits\,xs)),\ x * min\,(map\,prod\,(inits\,xs))) \qquad \text{(R3–5)}$$

$$= 1 \downarrow (x \geq 0 \rightarrow x * mxpi\,xs,\ x * mnpi\,xs)$$

ただし，補助関数

$$mnpi\,xs = min\,(map\,prod\,(inits\,xs))$$

を新たに導入した．$mnpi$ に関しても上と同様の変換を続けると，次の相互再帰定義が得られる．

$msp\,[\,] = 1$

$msp\,(x : xs) = x \geq 0 \rightarrow (1 \uparrow (x * mxpi\,xs)) \uparrow msp\,xs,$
$(1 \uparrow (x * mnpi\,xs)) \uparrow msp\,xs$

$mxpi\,[\,] = 1$

$mxpi(x : xs) = 1 \uparrow (x \geq 0 \rightarrow x * mxpi\,xs,\ x * mnpi\,xs)$

$mnpi\,[\,] = 1$

$mnpi(x : xs) = 1 \downarrow (x \geq 0 \rightarrow x * mnpi\,xs,\ x * mxpi\,xs)$

第二段階 (組化)

$msptup\,xs = (msp\,xs,\ mxpi\,xs,\ mnpi\,xs)$ として，重複した関数呼出しを除去する．結果は次の通り．

$msp\,xs = fst\,(msptup\,xs)$

$msptup\,[\,] = (1, 1, 1)$

$msptup\,(x : xs)$

$= x \geq 0 \rightarrow (a \uparrow u,\ a,\ 1 \downarrow q),\ (b \uparrow u,\ b,\ 1 \downarrow p)$

where $(u, v, w) = msptup\,xs,\ p = x * v,$
$q = x * w,\ a = 1 \uparrow p,\ b = 1 \uparrow q$

5　二次元問題への応用

本節では，FT 変換をより複雑な MSS2・MSP2 問題に適用し，その有効性を示す．初期プログラムを，図 5 に示す．スペースの関係上，変換の詳しい途中経過は省略するが，MSS2 問題については，

$$mxst\,xss = map\,(max \circ map\,sum2)\,(tops\,xss)$$

が，MSP2 問題については，

$$mxpt\,xss = map\,(max \circ map\,prod2)\,(tops\,xss)$$
$$mnpt\,xss = map\,(min \circ map\,prod2)\,(tops\,xss)$$

$mss2\,xss = fst\,(mss2tup\,xss)$

$mss2tup\,[\,] = (0, [0, \cdots, 0])$

$mss2tup\,(xs : xss) = (max\,p \uparrow u,\ p)$

where $(u, v) = mss2tup\,xss,$
$p = zip\,(\lambda(x, y)\,.\,x \uparrow (x + y))\,(map\,sum\,(segs\,xs))\,v$

$msp2\,xss = fst\,(msp2tup\,xss)$

$msp2tup\,[\,] = (1, [1, \cdots, 1], [1, \cdots, 1])$

$msp2tup\,(xs : xss) = (max\,p \uparrow u,\ p,\ q)$

where $(u, v, w) = msp2tup\,xss,\ m = map\,prod\,(segs\,xs)$
$p = zip3(\lambda(x, y, z)\,.\,x \geq 0 \rightarrow x \uparrow (x * y),\ x \uparrow (x * z))\,m\,v\,w$
$q = zip3(\lambda(x, y, z)\,.\,x \geq 0 \rightarrow x \downarrow (x * z),\ x \downarrow (x * y))\,m\,v\,w$

図 6　FT 変換による MSS2・MSP2 問題の最終結果

の二関数が，変換途中で補助関数として導入され，これらが組化の対象となる．図 6 に MSS2・MSP2 両問題に関する FT 変換の最終結果を示す．ここで $zip3$ は zip の自然な拡張である．

与えられる矩形の一辺を n とすると，初期プログラムは双方とも $O(n^6)$ であったが，FT 変換による融合・組化により $O(n^3)$ のプログラムが得られた．

6　結　論

本稿では，融合変換の後に組化変換を続けて施すことにより，中間データ構造を生成せず，同一のデータを重複して辿らないような，効率のよいプログラムを導出できることを示した．変換途中で出現した部分式に対して (必要に応じて) 補助関数を導入し，この補助関数の定義に対して融合操作を続けるために，漸次的な変換が可能である．この点において，時として深い洞察力を必要とする BMF と比較すると，より系統的な変換が可能となる．実際，MSP などいくつかの問題について，系統的な変換操作で効率のよいプログラムが導出された．

参考文献

[1] Bird, R.S. and Wadler, P.: Introduction to Functional Programming, Prentice–Hall, 1988.

[2] Bird, R.S.: Lecture Notes on Theory of Lists: *STOP Summer School on Constructive Algorithmics*, pp.1–25, 1989.

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

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

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