論　文

UDC 519.682.8:681.32.068

関数型言語ASL/Fとその最適化コンパイラ

正員　井上　克郎†　　正員　関　浩之†
正員　谷口　健一†　　正員　嵩　忠雄†

Functional Programming Language ASL/F and Its Optimizing Compiler

Katsuro INOUE†, Hiroyuki SEKI†, Kenichi TANIGUCHI† *and* Tadao KASAMI†, *Regular Members*

あらまし　本論文では，関数型プログラミング言語ASL／Fを定義し，通常の計算機上で効率良く実行するためのコンパイルや最適化の方法について述べる．また，これらの方法の有効性を調べるため，実際にコンパイラを試作し，いくつかの例プログラムを実行した．ここではその実行結果についても述べる．ASL／Fプログラムは，いくつかの関数定義文と評価すべき項（計算項）の記述からなり，またその意味は，項の書換えで簡明に定義される．最適化項目としては，定義関数の必須引数に対する先評価，共通部分項の重複計算の除去，配列等のソート（データタイプ）の大域化，Tail Recursionの除去，コンパイル時における項の書換え，を採用し，コンパイラで実現した．例プログラムの実行結果から，①これらの最適化は実行時間，使用領域を大きく減少させる，②同等なPASCALプログラムとほぼ同じ時間で実行できる，等がわかった．

1. まえがき

関数型言語は，一般に意味の定義が簡明で，検証が比較的容易にかつ厳密に行なうことができる等の特徴を持つが，通常の計算機上で効率良く実行するためのコンパイルや最適化の方法については，ほとんど研究されていない．

本稿では，簡潔な関数型言語ASL/F（a Functional language as a subset of an Algebraic Specification Language[6]）を定義し，そのコンパイル，最適化の方法を提案する．また，それらの方法の有効性を調べるために実際にコンパイラを試作し，いくつかの例プログラムを実行してみた．ここではその実行結果についても述べる．

ASL/Fはいわゆる計算可能関数を記述できる意味で万能であり，また，アルゴリズムをプログラム化するのも，従来の手続的言語と同じように普通に行える．

ASL/Fは，そのプログラムの意味の定義が，項の書換えで簡明に行なわれ，また，LISPにおけるリストやFP[1]における木のように，特定のデータ構造に依存もしていない．従ってASL/Fに任意のデータ構造を組込むことが，自然に行なえる．

ASL/Fコンパイラは，ASL/Fプログラムをアセンブリ言語等の手続的言語の目的プログラムに変換する．目的プログラムは，入力データを計算項中の各変数に代入し，得られた項の値を計算して出力する．本稿で述べるコンパイル及び最適化の方針としては，

① ここでは目的プログラムの実行効率の向上（実行時間や使用記憶領域を減らす）に注意を払い，目的プログラム自身の大きさ，コンパイルや最適化に要する時間については特に考慮しない．

② ここで扱う最適化は，主として，この種の言語特有なものに限定し，ソースプログラムそのものを解析の対象にする．従って通常の手続的言語で行なわれるような最適化（例えばレジスタ割付け，ループ内不変式の移動等）を目的プログラムに対して行なうことについては考慮しない．

† 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

2. ASL/Fの概略

2.1 シンタクス

2.1.1 ソート，関数及び項

整数，ブール等のデータ型のことをここではソートと呼ぶ．ASL/Fでは通常の言語で用いられる整数，実数，ブール，文字，文字列，（整数等の）配列，及びそれらの並び（tuple）等のソート（基本ソート）とそれらに関する基本関数が用意されている（表1に現段階の試作コンパイラで実現されている基本関数を示す）．n引数関数fがそれぞれソート$s_1, s_2, \cdots, s_n$の値を引数とし，ソートsの値を関数値とするならば，$f: s_1, s_2, \cdots, s_n \rightarrow s$と書く．

ソートsの定数（ソートsの元）それ自身をソートsの項と呼ぶ．また関数fが$f: s_1, \cdots, s_n \rightarrow s$であり，$t_1, \cdots, t_n$が各々ソート$s_1, \cdots, s_n$の項であるとき$f(t_1, \cdots, t_n)$もソート$s$の項と呼ぶ．

2.1.2 定義関数及びIF関数

ASL/Fには基本関数，IF関数，定義関数の三種の関数がある．各定義関数gに対し定義文が唯一つ存在し，$g(x_1, \cdots, x_n) == \tau_g$の形を持つ．ここで$g(x_1, \cdots, x_n)$を左辺，$\tau_g$を右辺（の項）という．$x_1, \cdots, x_n$は相異なる変数で$\tau_g$には変数としては，$x_1, \cdots, x_n$以外のものは現れない．

IF関数の関数値は，その第1引数の値がTRUEの時は第2引数の値，FALSEの時は第3引数の値である．任意のソートsについて関数名IF_sがあり，それぞれ次の公理を満たす．

$IF_s(TRUE, x_1, x_2) == x_1$

$IF_s(FALSE, x_1, x_2) == x_2$

ただしプログラムテキスト上には，IF_sの代わりに単にIFと略記できる．

2.1.3 ASL/Fプログラムの構造

図1は，配列として与えられた整数列を昇順に並べかえるクィックソートプログラムであり，Wirthによる図2のPASCALプログラム（入出力は省略してある）[4]と同一アルゴリズムを持つ．(1)はプログラム名の宣言で，(2)は配列に関する記述ARRAY（3つの引数を持つ）をこのプログラムに取込む宣言であり，ここでは実引数として，INT（配列要素のソートを整数とする），5000（要素数），ARY（この配列のソート名†）を持つ．(3)〜(10)は各定義関数の引数及び関数値のソートの宣言文，(11)〜(18)は各々の定義文である．(19)は，評価してその結果を出力すべき項（計算項）である．もし計算項が変数を含むならば，データとして入力された基本ソートの値が各変数に代入された後，評価される．

表1 試作コンパイラで現在実現されている基本関数

論理演算	AND, OR, XOR, NOT
算術演算	ADD, SUB, TIMES, DIV, MOD, NEG
整数の比較	EQ, NEQ, GT, GE, LT, LE
配列用演算	CONTENT, ASSIGN
並びに関する演算	<>, PR1, PR2, PR3, PR4, PR5, PR6, PR7, PR8, PR9

- AND, …, ADD, …, EQ, … 等は通常の言語で用いられる関数（演算）と同等の意味の関数（TIMESは乗算）．
- CONTENT(X, i)は配列Xの第i要素を表わす．ASSIGN(X, i, d)は配列Xの第i要素をdで置換えた配列．
- $<d_1, d_2, \cdots>$は$d_1, d_2, \cdots$を要素とする並びの値を表わす．PR1(Y), PR2(Y), …はそれぞれ並びYの第1要素，第2要素，…を表わす．

2.2 ASL/Fプログラムの意味

いま，PをあるASL/Fプログラム，$T(P)$をP中の各変数や関数，定数から構成されうる項の集合とす

```
(1) SPEC QUICKSORT ;
(2) INCLUDE ARRAY(INT, 5000, ARY) ;
(3) OP QSORT    : ARY, INT, INT                          -> ARY ;
(4)    SPRQSORT : ARY, INT, INT, INT, INT, INT -> ARY ;
(5)    LEFT     : ARY, INT, INT                          -> INT ;
(6)    RIGHT    : ARY, INT, INT                          -> INT ;
(7)    EXCH     : ARY, INT, INT                          -> ARY ;
(8)    MID      : INT, INT                               -> INT ;
(9)    INC      : INT                                    -> INT ;
(10)   DEC      : INT                                    -> INT ;
   AXIOM
(11) QSORT(X, I, J) == IF( GE(I, J), X,
        SPRQSORT( X, I, J, I, J, CONTENT(X, MID(I, J))) )  ;
(12) SPRQSORT(X, I, J, L, R, B) ==
       IF( LT(LEFT(X, L, B), RIGHT(X, R, B)),
         SPRQSORT( EXCH(X, LEFT(X, L, B), RIGHT(X, R, B)),
                   I,  J,
                   INC(LEFT(X, L, B)),  DEC(RIGHT(X, R, B)),
                   B  ),
         IF( EQ(LEFT(X, L, B), RIGHT(X, R, B)),
             QSORT( QSORT(X, I, DEC(RIGHT(X, R, B)) ),
                    INC(LEFT(X, L, B)),  J  ),
             QSORT( QSORT(X, I, RIGHT(X, R, B) ),
                    LEFT(X, L, B),  J  ) ) )                ;
(13) LEFT(X, L, B) ==
       IF( GE(CONTENT(X, L), B), L, LEFT(X, INC(L), B) )    ;
(14) RIGHT(X, R, B) ==
       IF( LE(CONTENT(X, R), B), R, RIGHT(X, DEC(R), B) )   ;
(15) EXCH(X, P1, P2) == ASSIGN(ASSIGN(X, P1,CONTENT(X, P2)),
                                P2, CONTENT(X, P1) )        ;
(16) MID(I, J) == DIV(ADD(I, J), 2)                         ;
(17) INC(I) == ADD(I, 1)                                    ;
(18) DEC(I) == SUB(I, 1)                                    ;
   END
(19) QSORT(X, 1, N)
```

```
SPRQSORT(X, I, J, L, R, B)  sorts the elements X[I],
X[I+1], ..., X[J-1] and X[J] of array X under the assumption
that X[k] ≤ B for each k , I ≤ k ≤ L
and  X[l] ≥ B for each l , R ≤ l ≤ J .
```

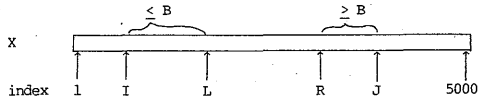

図1 ASL／Fによるクィックソートプログラム
Fig.1 Quicksort program in ASL/F.

† ソートARYの値は，サイズ5000の整数配列である．要素数や要素のソートが異なる配列に対しては，異なるソート名で宣言する．また，それらが同じであっても異なるソート名で複数個宣言してもよい．

```
PROCEDURE QUICKSORT ;
VAR  N,I : INTEGER ;

PROCEDURE SORT(L, R :INTEGER) ;
VAR I, J, M, W :INTEGER ;
BEGIN
  I:=L ;  J:=R ;  M:=X[(L+R) DIV 2 ] ;
  REPEAT
     WHILE X[I] < M DO I:=I+1 ;
     WHILE M < X[J] DO J:=J-1 ;
     IF I<=J THEN BEGIN
        W:=X[I] ; X[I]:=X[J] ; X[J]:=W ;
        I:=I+1 ; J:=J-1
     END
  UNTIL I>J ;
  IF L < J THEN SORT(L, J) ;
  IF I < R THEN SORT(I, R)
END ;

BEGIN  SORT(1, N)  END
```

図2　PASCALによるクィックソートプログラム
Fig. 2 Quicksort program in PASCAL.

る．P中の①各定義文，②各ソートに対するIFの公理，③各基本関数の値を決める公理（例えばADD(0, 0)==0，ADD(0, 1)==1，…，可算無限個ある）の各々を，左辺から右辺への書換え規則[7]（$t_i \to t_j$とかく）としたものの集合をRULE(P)と定める．RULE(P)中のある書換え規則を用いて項tの部分項を書換えると，tがt'になる（$t, t' \in T(P)$）とき，$t \Rightarrow t'$とかく．

ある項$t_1 \in T(P)$について，もし，① $t_1 \Rightarrow t_2$，$t_2 \Rightarrow t_3, \cdots, t_{m-1} \Rightarrow t_m$が成立ち，② $t_m \Rightarrow t_{m+1}$というt_{m+1}が存在せず，さらに，③t_mが定数であるならば，t_mをt_1のPにおける値と呼ぶ（又はt_1は値t_mを持つという）．RULE(P)は，Church-Rosserの性質を持つので[5]，項tが値を持つならばそれは書換え順によらず一意に定まる．

いま，変数$x_1, \cdots, x_{n_P}$を含む計算項を$t_P(x_1, \cdots, x_{n_P})$とし，$x_1, \cdots, x_{n_P}$に対する入力データを$D = <d_1, \cdots, d_{n_P}>$とする．$t_P(x_1, \cdots, x_{n_P})$の各変数$x_i$を$d_i$で置換えて得られる項$t_P(d_1, \cdots, d_{n_P})$の値を，$P$の$D$に対する計算値という．

3. 必須頂点列

3.1 項の木表現

項tを木で表わしたものをtree(t)=(V, A)（Vは頂点の集合，Aは（有向）辺の集合）とする．木の各頂点vには関数名，定数，又は変数名がラベル（label(v)で表わす）として付けられる．A中の各辺$e=(u_1, u_2)$には，u_1から見てu_2が第i子（即ち関数label(u_1)の第i引数がu_2に相当する）のとき，整数iが付けられる．

V中のある頂点vに対し，term(v)はvを根とする部分木に対応する（部分）項を表わす．（変数への代入σに対する）vの値とは，term(v)中に含まれる変数に代入σを行って得られた（変数を含まない）項σ(term(v))の値をいう．簡単化のため以降σを陽に書かない．“項の書換え”と同様に“木の書換え”という表現を以後自由に用いる．

3.2 書換え順序

IF関数をラベルとして持つ頂点に対しその第1子，また基本関数をラベルとして持つ頂点に対しその各子供，をそれぞれ必須子という．ある頂点とその必須子を結ぶ枝を必須枝という．いま，ある頂点v_1が必須子v_2を持つならば，（必須子の定義と公理の形より）v_2の値が計算されなければv_1の値は求まらない．またv_1が必須子でない子供v_2を持つならば，v_2をv_1より先に書換えると（v_1が値を持つにもかかわらず）値が得られなかったり，（v_1の値に影響を与えないような）無駄な書換えが生じる場合がある．

いまある木tree(t)=(V, A)中の頂点vから必須枝のみをたどって到達できる頂点の集合をNEED(v)とする（v自身も含む）．前述のように，vの値を得るにはNEED(v)の各頂点をいわゆる“葉から根へ”の順に，全て値に書換えねばならない．そこでNEED(v)中の各頂点の値の計算順を次の①，②のように定め，この計算順に頂点を並べたものを（vの）必須頂点列（Needed Node Sequence for v）といい，S_vで表わす．なおNEED(v)中の各頂点uには，uの必須子間の評価順（必須子集合上の全順序関係$\ll_u$）が指定されているとする．① w_kがw（$w, w_k \in$ NEED(v)）の一つの必須子で，wの他の（w_k以外の）全必須子（$\in$ NEED(v)）がすでに初期部分列中に現われているとき，w_kの直後はwである（“計算の断片”を少なくするため）．②必須子間の順序は評価順に従う（$w_i \ll_w w_j$ならばw_iはw_jよりも前（左）に現われる）．すなわち，vの必須頂点列は，NEED(v)が張る木を必須子間の評価順に従いつつ後行順走査（Postorder Traversal）[3]することにより得られる．

図3に必須頂点列の例を示す．この例では各頂点uの必須子間の評価順は，各々uの第i子v_i，第j子v_jに対し，$i > j$ならば$v_i \ll_u v_j$となっている（右優先）．（現段階の試作コンパイラでも，各木tree(t)（tは定義文右辺の項又は計算項）の各頂点に対し，同様な評価順を与えている．）必須頂点列は，以降の目的プログラム発生や最適化に用いる．

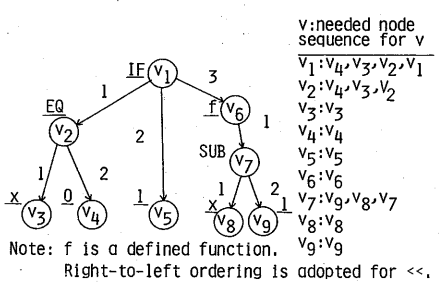

図3　必須頂点列の例
Fig.3 - Example of needed node sequences.

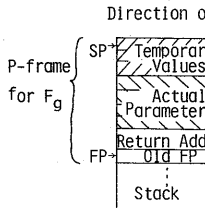

図4　親手続きF_g実行時のスタック
Fig.4 - Stack when master procedure F_g is active.

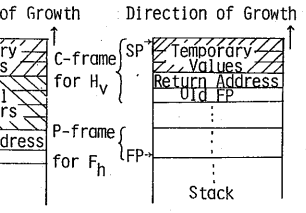

図5　（親手続きF_hの）子手続きH_v実行時のスタック
Fig.5 - Stack when slave procedure H_v (of procedure F_h) is active.

4. 目的プログラム

def(P)をP中の定義関数名の集合とする．議論が簡単になるよう，MAINという新しい“定義関数”を設け，計算項t_Pに対しMAIN($x_1, \cdots, x_{n_P}$)==t_P($x_1, \cdots, x_{n_P}$はt_P中に現われる変数）という“定義文”を仮定し，DEF(P)=def(P)∪{MAIN}とする．また，RIGHT(P)=$\{\tau_g \mid g \in DEF(P)\}$（$\tau_g$は$g$の定義文右辺の項を表わす）とする（すなわちRIGHT(P)はPの各定義文右辺の項及びt_Pの集合）．

4.1 関数手続き及び主手続き

目的プログラムは，1つの主手続き，及びいくつかの親手続きや子手続きから構成される．

親手続き：各$g \in DEF(P)$に対してgの定義文右辺の項τ_gの値を計算する（関数）手続き$F_g(l_1, \cdots, l_{n_g})$を設ける．$l_1, \cdots, l_{n_g}$は$g$の定義文左辺に現われる変数$x_1, \cdots, x_{n_g}$に対応する仮引数である．この手続きを親手続き（Master Procedure）と呼び，仮引数を省略してF_gと書くことがある．

子手続き：各tree(τ_g)($g \in DEF(P)$)中のラベルが定義関数eである頂点の各子供vに対し，vの値を求める（関数）手続きH_vを設け，これを（F_gの）子手続き（Slave Procedure）と呼ぶ．H_vはF_eの実行中vの値が必要になった時点で呼ばれる（定義関数の引数の遅延評価[Delayed Evaluation[2]]）．子手続きは，陽には仮引数を持たない．

主手続きは，次の(1)～(3)を実行する．いま計算項t_P中の変数を$x_1, \cdots, x_{n_P}$とする．(1)入力データ$d_1, \cdots, d_{n_P}$を読む．(2)$d_1, \cdots, d_{n_P}$を実引数として$F_{MAIN}(l_1, \cdots, l_{n_P})$を呼ぶ．(3)$F_{MAIN}(l_1, \cdots, l_{n_P})$の値を出力，印字する．

4.2 フレーム

各手続きが使用する一定の領域をフレームと呼ぶ．フレームにはPフレームとCフレームがあり，共通の一本のスタック上に取られる．

親手続きF_gの実行時には，図4のようにスタック最上に置かれたPフレームに対し，（**4.3**(a)～(e)で述べるような）値の読み書きが行なわれる．この時このPフレームの最上はスタックポインタSP，底はフレームポインタFPによって指される．Pフレーム内には，①現在実行中の手続きF_gを呼出した手続きの実行時のFPの値（前FP[old FP]という），②戻り番地，③実引数，④計算の途中結果の値（この個数は実行の進行とともに変わる）が蓄えられる．

一方ある親手続きF_hの子手続きH_vの実行時には，図5のようにスタック最上に置かれたCフレームに対し（**4.3**(a)～(d)で述べるような）読み書き，及びF_h実行時（H_vがある親手続き$F_{g'}$によって呼出されたならば$F_{g'}$を呼出したF_h，H_vがある子手続き$H_{u'}$によって呼出されたならば$H_{u'}$の親手続きを呼出したF_h）に用いたPフレーム中の実引数に対して（**4.3**(e)で述べるような）読出し，がそれぞれ行なわれる．この時，このCフレームの最上はSPによって，Pフレームの底はFPによって指されている．Cフレーム内には，①前FP，②戻り番地，③計算の途中結果の値（個数は実行の進行とともに変わる）が蓄えられる．

4.3 命令列

項τ_g($g \in DEF(P)$)の木表現tree(τ_g)の根をrとする．親手続きF_gは次のような命令列によって構成される（Gen-objについては後で述べる）．なお，sp，fpをそれぞれSP，FPの値（番地）とし，$\alpha\uparrow$をα番地の値とする．またX←βは，値βを㋑XがSPかFPならばポインタXに，㋺それ以外ならばX番地に代入することを意味する．ここでは説明を簡単にするため，フレーム中の各値はそれぞれ一語を占めるもの

とする（一つの番地に一つの値）．またここでは一切の最適化を行なわない命令列について述べる．

① Gen-obj(τ)により構成される命令列；この命令列実行完了時には，$sp=fp+m_g+2$（m_gはF_gの仮引数の個数）が成立ち，SPが指す番地にこの命令列の実行結果が格納されている．

② 当該親手続きからの復帰の為の命令†．即ち，
- FP←fp↑；FPを前FPの値にする．
- SP←($sp-m_g-2$)；SPはPフレーム底を指す．
- sp←($sp+m_g+2$)↑；①で求めた値（SPが指す番地よりm_g+2だけ上［スタックの伸びる方向］にある）をSPが指す番地に転送．
- GOTO($sp+1$)↑；戻り番地に復帰．

一方子手続きH_vの命令列は，

① Gen-obj(v)によって構成される命令列；この命令列実行完了時には，Cフレームは3語からなる（前FP［FPが指している］，戻り番地，実行結果［SPが指している］）．

② 当該子手続きからの復帰の為の命令．即ち，
- FP←fp↑；FPを前FPの値にする．
- SP←($sp-2$)；SPはCフレーム底を指す．
- sp←($sp+2$)↑；①で求めた値をSPが指す番地に転送．
- GOTO($sp+1$)↑；戻り番地に復帰．

uをtree(τ_g)中の頂点とし（$g\in$DEF(P)），$S_u=v_1v_2\cdots v_k$をuの必須頂点列とすると，Gen-obj(u)は命令列$I_{v_1}I_{v_2}\cdots I_{v_k}$を出力する．ここで$I_{v_i}$はlabel($v_i$)により次の様に定義される命令である．

(a) label(v_i)が基本関数であれば，I_{v_i}はその関数を実行する命令（例えばlabel(v_i)="ADD"ならば加算命令）．これらの命令実行に必要な値（引数）はすでに求められて（計算の途中結果の値として）フレーム最上にある（sp↑，($sp-1$)↑，… で参照できる）．また命令の実行結果も，引数の値を除去した後のフレーム最上に置かれる．例えば"ADD"ならば，($sp-1$)←(sp↑+($sp-1$)↑)；SP←($sp-1$)．

(b) label(v_i)が定数cならば，($sp+1$)←c；SP←($sp+1$)．

(c) label(v_i)がIF関数ならば，I_{v_i}は，v_iの第1子v_{i1}の値の判定を行ない（$I_{v_{i1}}$はI_{v_i}より先に実行されて求められており，v_{i1}の値はsp↑で参照できる），もしTRUEであればαの実行，FALSEならばβの実行を行なう命令列．ただしα，βは，v_iの第2子，第3子をv_{i2}，v_{i3}とすると，それぞれGen-obj(v_{i2})，Gen-obj(v_{i3})で定義される命令列である（v_{i2}，v_{i3}共にS_u中に含まれない）．

(d) label(v_i)が定義関数qならば，次の様な親手続きF_qの呼出しの命令．($sp+1$)←fp；($sp+2$)←戻り番地（$I_{v_{i+1}}$の番地）；($sp+3$)←$H_{v_{i1}}$；($sp+4$)←$H_{v_{i2}}$；…；($sp+m_q+2$)←$H_{v_{im_q}}$；FP←($sp+1$)；SP←($sp+m_q+2$)；GOTO F_q．ここで$H_{v_{i1}}$，$H_{v_{i2}}$，…，$H_{v_{im_q}}$は，それぞれv_iの各子供（すなわちqの各引数，個数はm_q個）の値を計算するF_gの子手続き（の入口番地）．

(e) lebel(v_i)が変数x_jならば，x_jに対応する（子手続きの入口番地である）実引数をH_{w_j}（($fp+j+1$)↑で参照できる）とすると，子手続きH_{w_j}を呼出す次の様な命令．($sp+1$)←fp；($sp+2$)←戻り番地（$I_{v_{i+1}}$の番地）；WK←($fp+j+1$)↑；FP←fp↑；SP←($sp+2$)；GOTO wk．ただしWKは値の一時退避用レジスタで，wkはその内容である．

5. 最適化

5.1 定義関数の必須引数の先評価

5.1.1 必須引数優先

ある項$g(t_1,\cdots,t_i,\cdots,t_m)$（$g$は定義関数）の値を求めるために，書換え順や$t_1,\cdots,t_m$の値によらず必ず$t_i$の値が必要なとき，$g$の第$i$引数は必須である（又は必須引数である）という．例えばgの定義を$g(x,y,z)$==IF(EQ(x,0),ADD(y,1),SUB(y,z))とすると，gの第1，第2引数は必須である（右辺の項τ_gの値を求めるにはEQ(x,0)の値が必要であり，更に$EQ(x,0)$の値を求めるにはxの値が必要である．またEQ(x,0)の真偽に従ってADD(y,1)，SUB(y,z)いずれの値を求めるとしてもyの値が必要である）．P中の定義関数の引数が必須であるための十分条件が知られている[8]．図1のプログラム中の定義関数の全引数はこの十分条件を満たしている（必須である）．

必須な引数の値を，親手続きF_gの呼出し前に計算し，その求まった値をF_gに渡すように計算順を変更する（必須引数優先［Needed-node-first］の計算順という）†．

† 目的プログラムの言語によっては命令列であるときもあるが，以下でも単に命令という．

† この計算順に対応する項の書換えの系列は，4.で述べた計算順に対応する項の書換えより書換え回数が増加したり，書換えが停止しなくなることはない．

この必須引数優先の計算順を実現した目的プログラムは，**4.**で述べた目的プログラムより次の理由により効率が良い．

(1) 必須な引数に対する子手続き呼出しが不要になり，手続き呼出し時や復帰時に必要なレジスタの退避，復帰，ポインタのセット等に要する手間が無くなる．

(2) 実行時の親手続きのネストの深さが小さくなり，最大スタック長が大幅に短くなる場合がある．

例えば v_2 を tree(τ_{g1})($g1 \in$ DEF(P)) 中の頂点とし，term(v_2)$=g2(g3(\cdots),\cdots)$($g2, g3 \in$ DEF(P))，v_2 の第1子を v_3 (即ち term(v_3)$=g3(\cdots)$) とする．そして親手続き F_{g1} 内で v_2 の値が必要であり，$g2$ の第1子は必須であるとする．**4.**で述べた計算順で τ_{g1} の値を求めると，実行する手続き及び使用するフレームは次の様になる (図6(1))．

(i) F_{g1} から親手続き F_{g2} を呼ぶ．その際 F_{g2} に対する P フレームをスタック上に取る．(ii) F_{g2} から (F_{g1} の) 子手続き H_{v_3} を呼び C フレームを取る．(iii) H_{v_3} から親手続き F_{g3} を呼び F_{g3} に対する P フレームを取る．

一方必須引数優先を実現した目的プログラムでは次の様になる (図6(2))．

(i) F_{g1} から F_{g3} を呼び F_{g3} に対する P フレームを取る．(ii) F_{g3} の実行完了後 F_{g3} に対する P フレームをスタック上から取除く．(iii) F_{g1} から F_{g2} を呼び F_{g2} に対する P フレームを取る．

5.1.2 必須引数優先の実現

必須引数優先を実現するために，**4.**で述べた目的プログラムに対し次の変更を行なう．

(1) ラベルが定義関数の頂点 u の必須子 u_j については子手続き H_{u_j} を作らない．

(2) **3.2**で述べた必須子に加え tree(t)($t \in$ RIGHT(P)) 中の頂点で，ラベルが定義関数の頂点に対し，必須な引数に対応する子供も必須子と呼ぶ．これらの必須子に対して**3.2**で述べた方法で必須頂点列を定義し，この必須頂点列を用いてGen-objにより命令列を**4.**と同様に作成する．ただし以下の1，2の変更を行なう．

1. label(v_i) が定義関数 g のとき，v_i の必須子の値 (すでに求っている) を F_g へ実引数として渡す．
2. label(v_i) が変数 x_j で x_j が必須な引数に対応する変数 (即ち値が実引数として渡されている) ならば，命令列 I_{v_i} は単に実引数を参照する命令．

5.2 その他の最適化

5.1で述べた最適化の他に，以下の(a)～(d)の最適化の具体的な実現方法を検討した．またそれらの最適化が行なえるための十分条件も調べた．これらの詳細は別に報告する予定である．

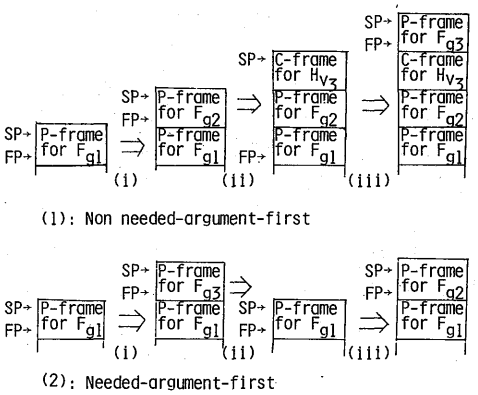

図6 必須引数優先による最大スタック長の減少
Fig.6 - Decrease of the maximum stack length by needed-argument-first.

(a) 共通部分項の重複計算の除去：木 tree(t)($t \in$ RIGHT(P)) 中の異なる頂点 v_1, v_2, …が term(v_1)$=$term(v_2)$=\cdots$であるとき，v_1, v_2, …の値を求める同一計算が重複して行なわれないように，一度求めた値をフレーム中に保存しておき，以降必要に応じてその保存した値の参照のみを行なう．

(b) 配列等のソートの大域化：配列等その値の保存に比較的大きな記憶域を要するソート s について，(もし可能ならば) ソート s の各値をフレーム中に蓄える代わりに，スタック外のある領域 W_s に蓄える．W_s はソート s のどのような値でも，少なくとも1つ格納できる．これによって領域使用量の減少や値の転送等に要する手間の減少をはかる．

(c) Tail-Recursionの除去：Tail-Recursive[3]に定義された定義関数は簡単なくり返し文で実行する．これにより実行時の最大スタック長が小さくなる．

(d) コンパイル時における書換え：RIGHT(P)の各項中の書換え可能な部分項の書換えを行ない，P中の各定義文右辺の項や計算項を書換えによって得られた項に置換えたプログラムに対し，他の最適化やコンパイルを行なう．(ただし書換えが停止するよう再帰的な定義関数の書換えは，行なわない．)

6. ASL/Fプログラムの例

上述のコンパイルや最適化の方法の有効性を調べるため，MELCOM COSMO900-Ⅱ(約6MIPS) 上

のオペレーティングシステムUTS/VSのもとで稼動するコンパイラを試作した†．このコンパイラは，ASL/Fプログラムをアセンブリ言語の目的プログラムに変換する．コンパイラ自身はPASCALで書かれ，約4000行の大きさ（構文解析，最適化，コード発生の3部からなる）で，設計，製作に7人月程度を要した．

6.1 最適化の効果

最適化の効果を知るために，次の問題を解くプログラムをASL/Fで作成した．(1)ソーティング（クィックソート）．(2)ソーティング（バブルソート）．(3)ハノイの塔．(4)自然対数の底 e の計算．(5)行列の積の計算．各プログラムは，特に実行効率等に注意を払わずに自然にアルゴリズムを記述したものである．

表2に各プログラムの実行時間，動的な領域使用量（最大スタック長）を示す．必須引数優先の最適化により，全プログラムの実行時間，領域使用量が**5.1.1**で述べた理由等により大きく減少している．

6.2 ASL/FプログラムとPASCALプログラムとの比較

表3に，**6.1**で述べた(1)〜(5)の各問題を，同じアルゴリズムを用いて，MELCOM PASCAL 8000で記述し，**6.1**と同一計算機上で実行した結果を示す．これらのPASCALプログラムは，FOR文，WHILE文等による繰返しや，あとで再び必要な中間結果の保存を行なう等，ある程度効率を考慮して書かれている．

各ASL/Fプログラムは，ほぼPASCALプログラムと同等の時間で実行可能であった．

コンパイルに要する時間は，例えばクィックソートの場合，ASL/Fからアセンブリ言語への変換（最適化に要する時間0.1秒も含む）に0.6秒，PASCALから機械語への変換に0.3秒であった．

6.3 ASL/Fインタプリタの作成

やや大きなASL/Fプログラムの例として，ASL/Fプログラム自身を直接実行するインタプリタ（ただし扱えるソートは整数，ブールのみ）を作成した．これは，文字コードの列として与えられた入力テキストを内部表現に変換する構文解析部と，項の書換えを（配列で模擬した）スタック上で行なう実行部から構成される．表4にこの各部の定義関数の数等を示す．

このインタプリタの設計から完成まで一人で2か月程度で，開発初期段階に存在した論理的な誤まりは数個であった．なおAckermann関数ACK(3,3)の値を求めるプログラムをこのインタプリタで実行すると3.6秒，コンパイラで実行すると0.01秒であった．

† 現段階のコンパイラでは，共通部分項の重複計算の除去の最適化を常に行なう．また配列の各ソートについて大域化できる場合にのみ目的プログラムを出力する（脚注1の後半の方法により配列の大域化ができるようにプログラムが書ける場合もある），5.で述べた他の最適化を行なうか否かは選択できる．この他冗長な転送命令の除去等の目的プログラムの簡単な改良を常に行なう．

表2 最適化の効果

実行時間（ミリ秒）

プログラム ＼ 最適化項目		①②	①②③	①②③④⑤
クィックソート	整数5000個	3200	990	430
バブルソート	整数 50個	53	18	6
	整数1000個	[20000]	7000	2400
ハノイの塔	10階	160	20	10
	15階	[5000]	630	350
e の計算	100桁	250	64	23
	1000桁	[14000]	3500	1300
行列の積	50×50	4700	1100	640

最大スタック長（語＝32ビット）

プログラム ＼ 最適化項目		①②	①②③	①②③④⑤
クィックソート	整数5000個	73700	49900	433
バブルソート	整数 50個	31600	225	17
	整数1000個	[1.2×10^7]	4030	17
ハノイの塔	10階	37800	194	162
	15階	[1.2×10^6]	285	242
e の計算	100桁	44300	525	20
	1000桁	[2.8×10^6]	3670	20
行列の積	50×50	44100	818	31

[ ] 内の値は推測値

最適化項目 ①共通部分項の重複計算の除去
②配列等ソースの大域化
③定義関数の必須引数の先評価
④ Tail-Recursion の除去
⑤コンパイル時における書換え

表3 PASCALプログラムとの比較

実行時間（ミリ秒）

プログラム		ASL/F	PASCAL
クィックソース	整数5000個	430*	320**
バブルソート	整数 50個	6	8
	整数1000個	2400	3200
ハノイの塔	10階	10	11
	15階	350	370
e の計算	100桁	23	28
	1000桁	1300	1500
行列の積	50×50	640	820

* 図1のプログラムの実行．
** 図2のプログラムの実行．

表4　ASL/Fインタプリタの大きさ

	定義関数の数	定義関数の引数の個数の最大	定義関数の引数の個数の平均	関数のネスト数の最大	関数のネスト数の平均	ソーステキスト上での行数
構文解析部	110	10	4	7	3	350
実行部	70	6	4	11	3	200

7. むすび

ここで述べた最適化の方法は，目的プログラムの実行時間や領域使用量を大幅に減らし，ここで挙げた例プログラムについては，ほぼPASCAL プログラムと同程度の時間で実行可能になった．

現段階のコンパイラでは，各頂点に与えられる必須子間の評価順は，それぞれ右優先であるが，**5.**の各最適化を行なえるための諸条件をできるだけ多く満たすような必須子間の評価順を見付けることができるなら，より実行効率を向上させることができるであろう．

なお，最適化の詳細については紙面の都合で省略したが，別稿で発表する予定である．

謝辞　種々有益な御討論，御助言頂いた杉山裕二博士に感謝する．また例プログラム作成に協力して頂いた現日本電気（株）増田勧氏に感謝する．

文　献

(1) Backus, J.: "Can Programming Be Liberated from the von Neumann Style ? A Functional Style and Its Algebra of Programs", Comm. ACM, 21, 8, pp. 613-641 (Aug. 1978).

(2) Henderson, P.: "Functional Programming, Application and Implementation", Prentice-Hall, pp. 214-231 (1980).

(3) Aho, A.V., Hopcroft, J.E. and Ullman, J.D.: "Data Structures and Algorithms", Addison-Wesley, pp. 66-67 and pp. 78-82 (1983).

(4) Wirth, N.: "Algorithms + Data Structurs= Programs", Prentice-Hall, pp. 76-82 (1976).

(5) 杉山，谷口，嵩："基底代数を前提とする代数的仕様"，信学論(D), J64-D, 4, pp. 324-331 (昭56-04).

(6) 杉山，谷口，嵩："代数的記述言語ASL／1－文法とその意味の定義－"，信学技報，AL82-62 (1982-11).

(7) 杉山，鈴木，谷口，嵩："あるクラスの項書換え系の効率のよい実行"，信学論(D), J65-D, 7, pp. 858-865 (昭57-07).

(8) 井上，関，杉山，嵩："関数的プログラミング言語ASL－Fのコンパイル時における最適化"，信学技報，EC82-18 (1982-06).

(9) 関，井上，杉山，嵩："関数的プログラミング言語ASL－Fコンパイラの作成"，情報処理学会第25回全国大会，1C－4，pp. 321-322 (1982-10).

(10) 井上，関，杉山，谷口，嵩："関数的プログラミング言語ASL－Fコンパイラ"，情報処理学会第26回全国大会，3D－7，pp. 13-14 (1983-03).

（昭和58年8月31日受付，11月2日再受付）