論　文

UDC 519.682.2:519.685.4

関数型言語ASL/Fのコンパイル時における最適化

正員　関　浩之[†]　　正員　井上　克郎[†*]
正員　谷口　健一[†]　　正員　嵩　忠雄[†]

Optimization of Functional Language ASL/F Programs

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

あらまし　関数型言語は，意味の定義が簡明である等の利点を持つが，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．本論文では，関数型言語ASL／Fプログラムを手続き的な目的プログラムにコンパイルする際の，いくつかの最適化の問題を定式化し，これらの最適化が行えるためのいくつかの十分条件を与える．最適化項目として，従来考察されていなかった「必須引数先評価」や「（配列等の）ソート（データタイプ）の大域化」を導入する．これらの他に，共通部分項の重複計算の除去，末端再帰の除去，コンパイル時の書換えを行う試作コンパイラを用いて例プログラムの実行効率を測定した結果，このような最適化により，目的プログラムの実行時間と動的記憶域使用量が大きく減少すること，Pascal 等の手続き的言語と同程度の時間で実行できることがわかった．

1. まえがき

関数型言語は，一般に意味の定義が簡明であり，従って，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．

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）の記述から構成される．プログラムの意味の定義は，項の書換えで簡明に行われる．

ASL／F コンパイラは，ASL／F プログラムを手続き的言語でかかれた目的プログラムに変換する．目的プログラムは，項の書換えを忠実に行うのではなく，各定義文に対応する関数手続きの呼び出しの繰り返しによりプログラムの値を求める．言語ASL／F 及びコンパイル法の概略については，文献(7)を参照されたい．

この言語の意味の定義により，プログラムの計算値を“失敗なく”求めるためには，各定義関数について，その実引数の値を求める前に関数手続きの呼び出しを行い，呼び出された手続き内で必要となった時点で実引数の計算を行えばよい．しかし，このような引数の遅延評価[(3)]は実行時のオーバヘッドが大きいので，引数値の計算を先に行ってもよい引数については，いわゆる“値渡し”を導入する（この手法を本文では“必須引数先評価”と呼ぶ）．また，値の格納に大きな記憶域が必要な配列等のソート（データタイプ）を含むプログラムを実用的な領域，時間で実行するために，（可能ならば）そのソートの値を一ヶ所の領域で保持する（これをソートの大域化と呼ぶ）．これらは，従来論じられていなかった新しい最適化の問題である．本論文では，これらの最適化の問題を定式化し，最適化が行えるための十分条件を与える．また，試作した最適化コンパイラ[(7)]を用いた測定データに基づいて，最適化の効果について述べる．

†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
*現在，Department of Information & Computer Sciences, University of Hawaii

2. ASL／Fの概略

ASL／Fには，整数，ブール，配列等のソート（データタイプ）と，それらのソートの定数，基本関数（四則演算，配列の要素を参照，変更する操作等に対応），及び次の２つの公理を満たすIF 関数が用意さ

れている．

IF(true, x_1, x_2)==x_1

IF(false, x_1, x_2)==x_2

また，プログラム中で新たに定義される関数を定義関数と呼ぶ．定義関数 g の定義文は，$g(x_1,\cdots,x_{n_g})==\tau_g$（$x_1,\cdots,x_{n_g}$ は異なる変数，項 τ_g 中には，$x_1,\cdots,x_{n_g}$ 以外の変数は現れない）の形のみ許す．定義文は，各定義関数につき，ちょうど1つのみ許す．

ASL／Fプログラムは，定義関数の引数及び関数値のソートの宣言，各関数定義文の記述及び目的とする計算を表す項（計算項）の記述からなる．

P をASL／Fプログラムとする．(1)基本関数の値を求める公理（ADD(0,0)==0，ADD(0,1)==1，…等，いわゆる関数定義表に相当），(2)IF関数の公理，(3)P 中の定義文，を左辺から右辺への書換え規則[5]とみなし，項 t を有限回書換えて t' が得られるとき，$t \xrightarrow{*} t'$ とかく．項 t に対し，(1) $t \xrightarrow{*} t'$ であり，(2) $t' \xrightarrow{*} t''$，$t' \neq t''$ なる t'' が存在せず，(3) t' が定数であるとき，t' を P における t の値と呼ぶ．項 t が値をもつならば，それは一意である[5]．

計算項 t_P が変数 $x_1,\cdots,x_{n_P}$ を含むとし，（P に対する）入力定数の組を $D=\langle d_1,\cdots,d_{n_P}\rangle$ とする．t_P の各変数 x_i を d_i で置換えて得られる項の値を（もし存在するならば）P の D に対する“計算値”という．

3. コンパイルの概略

MAINという新しい定義関数を導入し，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 に現れる異なる変数）なる定義文を設け，$DEF(P)=\{g \mid g$ は P 中の定義関数名$\}\cup\{$MAIN$\}$ と定める．

項 t を（有向）木で表したものを $tree(t)=(V,E)$（V は頂点の集合，E は（有向）枝の集合）と表す．木の各頂点 v には定数，関数名または変数名がラベル（label(v)で表す）として付けられる．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）項を表す．（変数への代入 σ に対する）v の値とは，term(v)中の変数に代入 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．簡単のため，以降 σ を陽にかかない．

以下では，定義関数 g の右辺を τ_g と表し，その木表現を $tree(\tau_g)=(V_g,E_g)$ とする．

IFをラベルとしてもつ頂点に対しその第1子，また，基本関数名をラベルとしてもつ頂点に対しその各子供をそれぞれ必須子という．また，ある頂点とその必須子を結ぶ枝を必須枝という．

P をASL／Fプログラムとし，v を V_g（$g\in$DEF(P)）中の頂点とする．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)で表す．v の値を得るにはNEED(v)の各頂点を“葉から根へ”の順に，全て値に書換えなければならない．NEED(v)中の各頂点を次の(1)，(2)のように並べたものを v の必須頂点列 $S(v)$[7]と呼ぶ．ただし，各 V_g（$g\in$DEF(P)）中の各頂点 u について，u の必須子間の（全）順序 $\ll_u$（u の必須子評価順と呼ぶ）が指定されているとする．

(1) $w,w'\in$NEED(v)，w' が w の必須子で，w の w' 以外の全必須子がすでに $S(v)$ 中の w' より前の初期部分列中に現れているとき，w' の直後は w である．

(2) $u\in$NEED(v)の2つの必須子 u_i，u_j について，$u_i \ll_u u_j$ ならば，u_i は u_j よりも前（左）に現れる．

目的プログラムは，(1)1つの主手続き，(2)各 $g\in$DEF(P)にたいして，τ_g の値を求める“親手続き”（F_g とかく），及び(3)各 τ_g 中の，ラベルが定義関数名である頂点の各子供 v に対し，v の値を求める（F_g の）“子手続き”（H_v とかく）からなる．目的プログラムは，必須子頂点列に対応する計算順をとるように構成される．親手続きへは，実引数として，変数に代入された項の値を計算する子手続き H_v の実行開始番地を渡し，呼び出された手続き内で，代入された項の値が必要になった時点で H_v を呼び出すようにする．各手続きが使用する一定の領域をフレームと呼ぶ．

頂点 v の値を求める命令列を $I(v)$ とする．$I(v)$ の詳細については，文献(7)を参照されたい．$I(v)$ の実行は，木の書換えにおいて，v の書換えが最初に行われてから，v の値が求まるまでの書換えに対応する．

4. 定義関数の引数の先評価

4.1 必須引数の定義とその先評価

一般に定義関数 g の引数の個数を n_g とし，その定義文を，$g(x_1,\cdots,x_{n_g})==\tau_g$ とする．t を任意の項，Xを変数名からなる任意の集合とする．$\sigma(t)$ が値をもつような任意の代入 σ に対し，少なくとも1つの $x\in$Xが存在して，$\sigma(t)$ の値を求めるために $\sigma(x)$ の値が必要であるとき，Xは t で必須であるという．また，$g\in$DEF(P)及び $1\leq k\leq n_g$ なる k について，$\{x_k\}$ が $g(x_1,\cdots,x_{n_g})$ で必須であるとき，g の第 k 引数は必須であるという．

g の第 i 引数が必須であるとき，その引数の値を親手続き F_g の呼び出し前に計算し，求まった値を F_g に

渡すように計算順を変更すれば（これを必須引数の先評価という），次の点で目的プログラムの実行効率が向上する．

(i) 必須な引数に対応する子手続きの呼び出し時における計算環境の切り換え等のオーバヘッドがなくなる．

(ii) 必須引数の先評価を行うと，最大スタック長が大幅に短くなる場合がある．例えば，定義文

$g(m,n)$==IF(EQ(n,0), m,
$g(g(\mathrm{ADD}(m,1), \mathrm{SUB}(n,1)), \mathrm{SUB}(n,1)))$

で定義される関数gの2つの引数はいずれも必須である．$g(i,j)$の値を求めるのに必要な最大スタック長は，必須引数の先評価を行えばオーダj，行わなければ2^jとなる．これは，それぞれの計算に対応する木の書換えでの，書換え途中での木の高さの最大値が，約j及び2^jであることを意味する．

(iii) 末端再帰的な形で定義された定義関数の各引数が必須であれば，それらの引数の先評価（これは通常の言語における“値渡し”に相当する）が行える．従って再帰の除去が行え，動的記憶域の使用量が減少する．また，ソートの大域化（**6.**で述べる）を考慮するとき，必須引数の先評価を行うという前提のもとでは，一般により多くのソートが大域化可能となる．

PをASL/Fプログラム，$g\in \mathrm{DEF}(P)$とする．集合$\nu_g \subseteq \{1,2,\cdots,n_g\}$が，

「$i\in\nu_g$ならば，gの第i引数は必須である」

を満たすとき，ν_gをgの必須引数指定という．各$g\in \mathrm{DEF}(P)$に対して，gの必須引数指定を定めたものをPの必須引数指定といい，ν_Pとかく．

プログラムPに対し，必須引数指定ν_Pが与えられているとする．uを，V_g（$g\in\mathrm{DEF}(P)$）中，ラベルが定義関数名の任意の頂点とし，$g'=\mathrm{label}(u)$とする．$i\in\nu_{g'}$であるようなuの第i子もuの必須子と呼ぶことにより，**3.**の必須子を（ν_Pに従って）拡張する．またこれにより，NEED(v)，必須子評価順$\ll_u$，必須頂点列$S(v)$のそれぞれを拡張する．プログラムPの必須子評価順全体をPの必須子評価順と呼び，$\ll_P$とかく．$S(v)$に対応する（必須引数の先評価を行う）Pの目的プログラムを，$\nu_P, \ll_P$に従うPの目的プログラムと呼ぶ．

4.2 必須引数の求め方

ここでは，指定された定義関数gの指定された引数が必須であるための十分条件を与える．尚この条件は，後述する仮定のもとで必須であるための必要条件ともなっている．

N_kで自然数の部分集合$\{1,2,\cdots,k\}$を表す．また自然数の部分集合Iに対し，変数名の部分集合X_Iを$X_I=\{x_i \mid i\in I\}$と定める．まず，次のようなブール型の変数を導入する．

・各$g\in\mathrm{DEF}(P)$，各$I\subseteq N_{n_g}$（$I\neq\phi$）に対して，
$Z[g,I]$（X_Iが$g(x_1,\cdots,x_{n_g})$で必須であることを表す）

これらの変数に対し，次のような方程式をたてる：
各$f\in\mathrm{DEF}(P)$，各$A\subseteq N_{n_f}$（$A\neq\phi$）に対して，
$Z[f,A]=Y[\tau_f, X_A]$
ここで$Y[t,X]$（tはあるτ_g（$g\in\mathrm{DEF}(P)$）の部分項，Xは変数名の部分集合）は，「Xがtで必須である」ことを表す論理式であり，以下のように定義される．

(1) tが定数のとき，$Y[t,X]=\mathrm{false}$
(2) tが変数xで$x\in X$のとき，$Y[t,X]=\mathrm{true}$
(3) tが変数xで$x\notin X$のとき，$Y[t,X]=\mathrm{false}$
(4) $t=p(t_1,\cdots,t_{n_p})$でpが基本関数のとき，
$Y[t,X]=Y[t_1,X]\vee\cdots\vee Y[t_{n_p},X]$（$X$が$t_1,\cdots,t_{n_p}$のいずれかで必須ならば，$X$は$t$で必須）
(5) $t=\mathrm{IF}(t_1,t_2,t_3)$のとき，$Y[t,X]=Y[t_1,X]\vee\{Y[t_2,X]\wedge Y[t_3,X]\}$（$X$が$t_1$で必須または$t_2$，$t_3$双方で必須ならば，$X$は$t$で必須）
(6) $t=h(t_1,\cdots,t_{n_h})$でhが定義関数のとき，

$$Y[t,X]=\bigvee_{\substack{A\subseteq N_{n_h}\\ A\neq\phi}}\{Z[h,A]\wedge\bigwedge_{i\in A}Y[t_i,X]\}$$

（ある空でない$A\subseteq N_{n_h}$が存在してX_Aが項$h(x_1,\cdots,x_{n_h})$で必須であり各$i\in A$についてXがt_iで必須ならば，Xはtで必須）

プログラムPに対し，上述のすべての方程式の集合からなる連立方程式をE_Pとかく．E_Pの1つの解をSとする．E_Pの解Sにおける$Z[g,I]$の値を$S[g,I]$とかく．また，各$Y[t,X]$に対し，$Y[t,X]$に現れる変数$Z[f,A]$に$S[f,A]$を代入し，簡単化して得られる値を$S(Y[t,X])$とかく．このとき次の性質が成り立つ．

［定理1］ SをE_Pの任意の解とする．$S[g,\{k\}]=\mathrm{true}$ならば，gの第k引数は必須である（証明は文献(6)，(8)参照）．■

$\geqslant$を$\mathrm{true}\geqslant\mathrm{false}$なる$\{\mathrm{true},\mathrm{false}\}$上の順序関係とする．$a\geqslant b$または$a=b$のとき$a\geqq b$とかく．解$MX$が，他の任意の解$S$に対し，「すべての$g\in\mathrm{DEF}(P)$，$I\subseteq N_{n_g}$（$I\neq\phi$）に対し，$MX[g,I]\geqq S[g,I]$」を満たすとき，$MX$を$E_P$の最大の解という．

$Y[t, X]$の定義から，E_Pの各方程式の右辺は$\geqq$に関して各変数について単調である（すなわち，解S, S'に対し，$Y[t, X]$に現れる各$Z[g, I]$について$S[g, I] \geqq S'[g, I]$ならば，$S(Y[t, X]) \geqq S'(Y[t, X])$が成り立つ）．従って，$E_P$は常に（$\geqq$に関して）最大の解をもつ．

プログラムPにおいて，(1)項$\mathrm{IF}(t_1, t_2, t_3)$が与えられたとき，$\sigma_1(t_1)$，$\sigma_2(t_1)$の値がそれぞれtrue，falseとなるような代入σ_1，σ_2が存在するか否か，(2)プログラム中に“冗長な”IF判定が存在する（例えば，項tに2つの部分項$t_0 = \mathrm{IF}(t_1, t_2, t_3)$，$t_0' = \mathrm{IF}(t_1', t_2', t_3')$が存在して，$\sigma(t_1)$の値がtrueとなる任意の代入$\sigma$に対し，$\sigma(t_1')$の値もtrueとなる）か否か，(3)項$t$が与えられたとき，$\sigma(t)$が値をもつような代入$\sigma$が存在するか否かを決める問題はいずれも一般には決定不能である．そこで次の仮定を考える．

（仮定1） 次の(i)，(ii)がともに成り立つ．

(i) 各τ_g（$g \in \mathrm{DEF}(P)$）の任意の部分項t_0及び，$t_0 \xrightarrow{*} t$なる任意のtに対し，次の条件が成り立つ：$\mathrm{tree}(t) = (V, E)$とし，ラベルがIFである$V$の各頂点の第1子に対し，trueまたはfalseを任意に割り当てる．そして，値を割り当てられた頂点のうち次の(イ)，(ロ)を満たすすべての頂点vからなる集合をV'とする．

(イ) vは定義関数名をラベルにもつ頂点を真の祖先にもたない．

(ロ) vの任意の真の祖先rに対し，rのラベルがIFであれば，vはrの第1子の子孫であるか，または，rの第1子にはtrue（false）が割り当てられておりvはrの第2(3)子の子孫である．

このとき，任意のtrue, falseの割り当てに対し，ある代入σが存在して，$v \in V'$ならば，$\sigma(\mathrm{term}(v))$の値はvに割り当てた値に等しい．

(ii) 各τ_g（$g \in \mathrm{DEF}(P)$）の任意の部分項tに対し，$\sigma'(t)$が値をもつような代入σ'が少なくとも1つ存在する． ■

仮定1の(i)は「各τ_g（$g \in \mathrm{DEF}(P)$）の任意の部分項t_0, $t_0 \xrightarrow{*} t$なる任意のtに対し，t中のIF関数の“条件判定の起こり方”の可能な組み合わせ†のそれぞれについて，そのような条件判定を引き起こすような代入σが存在し，$\sigma(t)$は値をもつ．」ということであり，その直観的な意味は，プログラム中に“冗長な”条件判定が全く存在しないということである．

［定理2］ E_Pの最大の解をMXとする．仮定1のもとで，定義関数gの第k引数が必須ならば，$MX[g, \{k\}] = \mathrm{true}$となる（証明は文献(8)参照）． ■

必須引数は，文献(4)，(5)における needed redex に相当する（ただし，文献(4)，(5)で議論されている書換え規則の左辺は，本論文の場合よりも一般的な形をもつ）．そこでは，左辺の情報のみを用いて needed redexを求めているが，本稿では，右辺も考慮に入れることにより，上述の条件を導くことができた．

次に，E_Pの最大の解を求めるアルゴリズムの概略を示す．すべての定義関数の引数の個数が，ある定数以下であると仮定し†，$n = \sum_{g \in \mathrm{DEF}(P)}$（$\mathrm{tree}(\tau_g)$中の頂点の個数）††とすると，このアルゴリズムの時間計算量は$O(n)$である．

まず，最大解を求める基本的方針を述べる．項と変数名の集合の対の集合$TX(P)$を$TX(P) = \{ <t, X> |$ ある$g \in \mathrm{DEF}(P)$があって，tはτ_gの部分項，Xは$\{x_i \mid i \in N_{n_g}\}$の部分集合$\}$と定め，$TX(P)$の部分集合の列$FC_0$, FC_1, …を次のように定義する．

(1) FC_0は，$Y[t, X]$の定義中の(1)～(3)のいずれかにより$Y[t, X]$の値が恒等的にfalseであることがわかる$<t, X>$全体の集合である．すなわち，

$FC_0 = \{ <t, X> | <t, X> \in TX(P)$，$t$は定数であるか，または，$t$が変数$x$で$x \notin X\}$

(2) $n \geqq 0$のとき，$FC_{n+1} = FC_n \cup \mathrm{New\_}FC$

ここで，$\mathrm{New_}FC$は，FC_nと$Y[t, X]$の定義により，$Y[t, X]$の値がただちにfalseとわかる$<t, X>$の集合であり，（$Y[t, X]$の定義式の形から）以下の(i)～(iii)のいずれかを満たす$TX(P)$の$<t, X>$全体からなる集合である．

(i) $t = p(t_1, \cdots, t_{n_p})$，$p$が基本関数で，すべての$i$（$1 \leqq i \leqq n_p$）に対し，$<t_i, X> \in FC_n$

(ii) $t = \mathrm{IF}(t_1, t_2, t_3)$で，$t_1 \in FC_n$かつ（$t_2 \in FC_n$または$t_3 \in FC_n$）

(iii) $t = f(t_1, \cdots, t_{n_f})$，$f$が定義関数で，$N_{n_f}$のすべての空でない部分集合$A$に対し，(イ)$<\tau_f, X_A> \in$

† 例えば項$t = \mathrm{IF}(t_1, t_2, t_3)$と代入$\sigma$に対し$\sigma(t_1)$の値がtrueのときは，$t_3$（else部）内の条件判定は考慮しない．また，$f$の定義文を$f(x, y) == x$とすると，$t = f(\mathrm{IF}(t_1, t_2, t_3), \mathrm{IF}(t_4, t_5, t_6))$は$t' = \mathrm{IF}(t_1, t_2, t_3)$に書換えられる．従って$\mathrm{IF}(t_1, t_2, t_3)$の条件判定は（$t'$で）考慮するが，（$t$の値を求めるために常に不要である）$\mathrm{IF}(t_4, t_5, t_6)$の条件判定は考慮しない．

† 通常，定義関数の引数の個数は，プログラムサイズには依存しないと考えられる．

†† nはソースプログラムの大きさとみなせる．

FC_n，または(ロ)少なくとも1つの $i \in A$ が存在して，
$< t_i, X > \in FC_n$ ■

FC_n の定義から，(i)任意の $k \geqq 0$ に対し，$FC_k \subseteq FC_{k+1}$，(ii)ある n が存在して，任意の k に対し，$FC_n = FC_{n+k}$，が成り立つ．(ii)を満たす最小の n を $\bar{n}$ とすると，次の性質が成り立つ．

（性質4.1） E_P の最大の解を MX とすると，$MX(Y[t, X]) = \text{false}$ となるときかつそのときのみ $< t, X > \in FC_{\bar{n}}$． ■

性質4.1より，E_P の最大の解を求めるには，各 $g \in \text{DEF}(P)$ 及び各 $I \subseteq N_{n_g}$（$I \neq \phi$）に対し，$< \tau_f, I > \in FC_{\bar{n}}$ であるか否かを判定すればよい．基本的には FC_k の定義に従って，$FC_0, FC_1, \cdots$ を順に求めていく方法をとる．この際，毎回 $Y[t, X]$ の値を（最初から）計算し直すことを避けるため，既に $Y[t', X']$ の値が false となることがわかっている $< t', X' >$ に対しては，値が false に確定したことを記憶しておき，二度以上 $Y[t', X']$ の計算は行わないようにする．実際には，$\text{tree}(\tau_g)$（$g \in \text{DEF}(P)$）中の各頂点 v に対応して領域 $v.Y[I]$（$I \subseteq N_{n_g}$，$I \neq \phi$）を設け（初期値は true），$Y[\text{term}(v), X_I]$ の値が false に確定したら，$v.Y[I]$ を false にセットする．

計算時間 $O(n)$ のアルゴリズムの概略は，次のようになる．新たに $Y[\text{term}(v), X_I]$ の値が false に確定した $< v, I >$ の集合を f_set とする．

(I) 各 $Z[g, I]$ の値を true にする．

(II) f_set を FC_0 に対応するように初期化する．

(III) f_set が空になるまで，以下の操作を繰り返す．f_set から任意に1つの $< v, I >$ を選び，$< v, I >$ を f_set から取り除き，次の(i)～(iii)を行う．

(i)各 v の子孫（v を含む）で，$v.Y[I] = \text{true}$ であるようなすべての u に対し，$u.Y[I]$ の値を false にセットし，(ii)（$Y[\text{term}(v), X_I] = \text{false}$ となったために）v の親 r について $Y[\text{term}(r), X_I] = \text{false}$ となるかどうかを調べる†．もし false ならば，$< r, I >$ を f_set に追加する．(iii) v がある τ_f（$f \in \text{DEF}(P)$）の根であれば，$Z[f, I]$ の値を false とし，f をラベルにもつようなすべての頂点 w（このような w はあらかじめ求めておくことができる）に対しても(ii)と同様の操作を行う．

(IV) 各 $Z[g, I]$ の値を出力する． ■

さて，$Y[t, X]$ の定義右辺において，“OR” 演算の一部を取り除き，かつどの方程式の右辺にも現れない変数を左辺にもつ方程式を取り除いた連立方程式についても定理1は成り立つ（すなわち $Z[g, \{k\}] = \text{true}$ ならば，g の第 k 引数は必須である）．例えば，$Y[t, X]$ の定義中の(6)において（すなわち $t = h(t_1, \cdots, t_{n_h})$ で h が定義関数のとき），A を元の個数が1個のものに限った連立方程式 E'_P についても定理1は成り立つ．P に現れる関数の引数の個数の最大値を m とすると，E'_P の最大解を上と同様の方法で求めるアルゴリズムの時間計算量は $O(m \cdot n)$ となる．現有の試作コンパイラでは，E'_P の最大解を求めることにより必須引数の検出を行っているが，文献(7)の例プログラムについては，各定義関数に対して，必須引数をすべて検出することができた．

5. 共通部分項の重複計算の除去

V_g（$g \in \text{DEF}(P)$）中の頂点の部分集合 $\{v_1, \cdots, v_m\}$（$m \geqq 2$）が，

(1) $\text{term}(v_1) = \cdots = \text{term}(v_m)$†，(2) $v_1, \cdots, v_m$ 以外の V_g 中の各頂点 u に対し，$\text{term}(u) \neq \text{term}(v_1)$，(3) $v_1, \cdots, v_m$ の各々の親を $u_1, \cdots, u_m$ とするとき，$\text{term}(u_i) \neq \text{term}(u_j)$ なる i, j（$i \neq j$，$1 \leqq i \leqq m$，$1 \leqq j \leqq m$）が少なくとも1組存在する．

を満たすとき，$\{v_1, \cdots, v_m\}$ を共通頂点集合という．また，共通頂点集合の各頂点を共通頂点という．

3.で述べた目的プログラムでは，V_g（$g \in \text{DEF}(P)$）内の異なる頂点 u, v について，$\text{term}(u) = \text{term}(v)$ であっても u, v の値を求める同一の計算が（ラベルが同一変数名のときの子手続き呼び出しによる計算も含め）重複して行われる場合がある．これを避けるため，各共通頂点 v に対し $I(v)$ を次のように一部変更する．

(1) V_g 中の各共通頂点集合 C に対応して，F_g に対するフレーム内に領域 a_C，b_C を設ける．ここで a_C は C に属する頂点の値を保存する領域，b_C は値がすでに求まって a_C に保存されている（真）か否（偽）かを表すフラグである（初期値は偽）．

(2) $I(v)$ の実行前に，フラグ b_C を調べ，(イ)「偽ならば $I(v)$ を実行し，その結果を a_C に保存する．そして b_C を真にセットする」，(ロ)「真ならば $I(v)$ を実行せずに a_C の参照のみを行う」．

0_P を，上述の重複計算除去を行った（ある ν_P，$\ll_P$ に従う）P の目的プログラムとする．C を共通頂点集

† このとき，r の各子供 v に対する $Y[\text{term}(v), X_I]$ の値としては，$v.Y[I]$ の値を用いる．

† $\text{term}(v_1)$ が変数の場合も含む．

合とし，$v_i \in C$とする．0_Pの実行中，$I(v_i)$におけるフラグ判定の結果が(i)常に偽である（すなわち，$I(v_i)$のどの実行開始でも，いずれの$v_k \in C$の値も求まっていない）か，(ii)常に真である（すなわち$I(v_i)$のどの実行開始時でも，それぞれ，ある$v_k \in C$があって，v_kの値が求まっている）ことがコンパイル時にわかれば，フラグの判定を省略することができる．そのための十分条件が得られている[8]．

6. ソートの大域化

6.1 ソートの大域化可能性の定義

項を表現するために，唯一の根をもったラベル付き連結有向アサイクリックグラフ（以下dagと呼ぶ）を用いる[5]．dagにおいては，根以外の頂点は一般に入射枝を複数本もつ．以下では，ラベルが変数名である頂点u, vに対し，label(u) = label(v)ならば，$u=v$であるとする．

木表現の場合と同様，dag dの各頂点vに対し，vを根とする部分グラフの表す項を term(v)で表す．特に，dag dの任意の頂点u, vに対し，

term(u) = term(v) ならば $u=v$

が成り立つとき，dを項t = term(r)（rはdの根）のdag表現と呼び，d = dag(t)とかく．すなわちdag(t)は，tree(t)において，同一の部分項を表す部分木をすべて1つにまとめたものである．tree(t)における共通頂点集合は，dag(t)においては入射枝を2本以上もつ頂点に対応し，根以外の共通頂点でない頂点は入射枝が1本である頂点に対応する．dag$(t)=(V^d, E^d)$, tree$(t)=(V, E)$と各$v \in V^d$に対し，TV(v)をterm(v)と同一の項を表すtree(t)の部分木の根の集合とする．すなわち，TV$(v)=\{u \mid$ term(u) = term(v), $u \in V\}$．以下では，各定義関数gの右辺τ_gのdag表現を，dag$(\tau_g)=(V_g^d, E_g^d)$とする．

プログラムPに対し，Pの必須引数指定ν_P及び必須子評価順$\ll_P$が与えられているとする．任意の入力D，及び各$g \in$ DEF(P)に対し，次の条件が成り立つとき，ソートsはPにおいて（ν_P, $\ll_P$について）大域化可能であるという．

「値のソートがsである各頂点$v \in V_g^d$，uの値を求める計算†の実行中にvの値を参照する各頂点$u \in V_g^d$，及び，wの値を求める計算の実行中にソートsの値を生成するv以外の各頂点$w \in V_g^d$に対し，vの値が求まってから，uの値の計算の実行中vの値の参照が行われるまでの間に，wの値を求めるためにソートsの値が生成されることはない．」

ソートsが大域化可能であれば，Pの目的プログラム0_Pを次のように修正しても，計算値は正しく求まる．また，このように目的プログラムを修正することを，「sを大域化する」という．

(1) ソートsのどのような値でも蓄えることのできる領域W_sをコンパイル時に“静的”に割当て（スタック外に確保し），スタック中にはソートsの値は置かない．

(2) sの値を参照する各命令は，W_sからその値を読み，sの値を生成する各命令は，W_sにその値を格納する．計算項に対する入力定数でソートsのものがあれば，その値もW_sに格納する．

ソートsが大域化可能となるように$\ll_P$を定めることも問題であるが，以下では$\ll_P$が与えられているとする．

6.2 大域化可能であるための十分条件

dag $d=(V^d, E^d)$において，ラベルが定義関数名gである頂点vにgの定義文右辺を“埋め込ん”だものをd'とするとき，d'はdより埋め込みにより得られるという．すなわち，d'は次のようにしてdより作られる．

(i) dag(τ_g)と同形のdag d_gを作る．

(ii) d中の枝(w, v)をすべて(w, r)（rはd_gの根）でおきかえる．

(iii) gの第i引数に対応する変数名をx_iとする．各x_iについて，x_iをラベルにもつd_g中の頂点v'を終点とするd_g中の枝(w, v')をすべて(w, v_i)（v_iはvの第i子）でおきかえる．

dag(τ_{MAIN})からn回の埋め込みで得られるdagを$d^n=(V^n, E^n)$とする．入力Dに対し，dag $d^n(s, D)$を$d^n(s, D)=(V^n(s, D), E^n(s, D))$とする．ここで，$V^n(s, D)$は，$D$に対してその値を求める計算が実行される$V^n$中の頂点で値のソートが$s$であるもの全体からなる集合であり，$E^n(s, D)=\{(u, v) \mid u, v \in V^n(s, D), (u, v) \in E^n\}$である．いま次の条件を考える．

（条件6.1） 各n（≥ 0），任意の入力D，及び各$d^n(s, D)$に対し，次の(1), (2)がともに成り立つ．

(1) $d^n(s, D)$が1本の単純な有向道になっている（このとき，$V^n(s, D)$の頂点をこの道に沿って並べ

† dag中の頂点uに対し，TV(u)中の頂点で，その値を求める命令列$I(\cdot)$が最初に実行されるものをu'とすれば，uの値を求める計算の実行とは，$I(u')$の実行を意味する．

たものを（根に近い方から）$v_1, v_2, \cdots, v_k$とする）．

(2) 各$(v_i, v_{i+1}) \in E^n(s, D)$に対し，(i) v_{i+1}がv_iの必須子であるか，または(ii) v_iのラベルがIFで，v_{i+1}がv_iの第2子または第3子である． ■

条件6.1が成り立つならば，$V^n(s, D)$に属する頂点の値は，$v_k, v_{k-1}, \cdots, v_1$の順に求まっていく．従って，大域化可能性の定義において，頂点wとしては，vの親†でありかつwの計算が行われるもの（上の仮定からこのようなwは高々1つしか存在しない）のみを考慮すればよい．条件6.1の（入力データDに依存しない形の）十分条件を考えると次の性質が成り立つ．

（性質6.1） 次の条件が成り立つならばソートsは大域化可能である．

（条件6.2） 任意のn（$\geqq 0$）に対し，次の[1]～[3]のすべてが成り立つ．

[1] $v \in V^n$の値のソートがsであり，vが値のソートがsである子供v'をもつならば，(i) v'はvの必須子であるか，(ii) label(v)＝IFで，v'はvの第2子または第3子である（ソートsの値は“葉から根へ”の順に求まっていく）．

[2] V^n中に，値のソートがsで，ソートsの引数をもたない頂点は高々1つしか存在しない．

[3] $v \in V^n$を値のソートがsである任意の頂点とする．vの異なる親u，wが存在して，wの値のソートがsならば，次の[3a]，[3b]のうち，少なくとも一方が成り立つ．

[3a] u, wのうち，高々一方の計算しか行われない．

[3b] uの値のソートはsでなく，uの値を求める計算の終了はwの値を求める計算の開始よりも先である．（証明略） ■

次に，（条件6.2）が成り立つための（各dag(τ_g)（$g \in$ DEF(P)）に対する）十分条件を述べる．

ソートs，及び各$g \in$ DEF(P)に対し，（$I(v)$の実行開始から終了までの間に，ソートsの値を生成しうる頂点vの集合）gen(g, s)を，次の(1)，(2)を満たすV_g^dの最小の部分集合とする．

(1) $v \in V_g^d$の値のソートがsで，そのラベルが基本関数名または定義関数名ならば，$v \in$ gen(g, s)

(2) $v \in V_g^d$のラベルlabel(v)が定義関数名で，gen(label(v), s)が空でなければ，$v \in$ gen(g, s) ■

u, vをV_g（$g \in$ DEF(P)）中の異なる頂点とする．任意の入力Dに対し，$I(u)$，$I(v)$のうち，一方が実行されるならば他方は実行されないとき，$u \oplus v$とかく．また，u，vに対し，任意の入力Dについて「Dに対し，もし$I(u)$，$I(v)$がともに実行されるならば，$I(u)$の実行開始は$I(v)$のそれよりも先である」とき，$u \prec v$とかく．$u \oplus v$が成り立つための十分条件，$u \prec v$が成り立つための十分条件については**6.3**で述べる．

（性質6.2） 各$g \in$ DEF(P)に対し，次の[1]～[3]のすべてが成り立つならば，条件6.2が成り立つ．従って，ソートsは大域化可能である．

[1] $v \in V_n^d$の値のソートがsであり，vが値のソートがsである子供v'をもつならば，(i) v'はvの必須子であるか，(ii) label(v)＝IFで，v'はvの第2子または第3子である．

[2] gen(g, s)中には，値のソートがsで，ソートsの引数をもたない頂点は高々1つしか存在しない．

[3] $v \in V_g^d$を値のソートがsである頂点とする．vの異なる親u，wが存在して，$w \in$ gen(g, s)ならば，次の[3a]，[3b]のうち，少なくとも一方が成り立つ．

[3a] 各$u' \in$ TV(u)，$w' \in$ TV(w)に対し$u' \oplus w'$が成り立つ．

[3b] 次の(イ)～(ハ)すべてが成り立つ．

(イ) $u \notin$ gen(g, s)．(ロ) uはwの祖先ではない．(ハ) TV(u)，TV(w)に属する頂点のうち，値を求める命令列$I(\cdot)$が最初に実行されうるものをそれぞれu'，w'とすると，このようなu'，w'の任意の組に対し，$u' \prec w'$が成り立つ（vの値の参照が行われてから，vの値の変更が行われる）．（証明略） ■

ソートsの引数を2つ以上もつIF関数以外の関数が存在すると（sの状態変更の道筋が2つ以上存在する可能性があるため），上の十分条件が成り立たない場合が多い．しかし，ソートsの引数を2個以上もつ関数が存在するとき，そのそれぞれに異なるソート名をつける（これは，手続き的言語のプログラムにおいて，同一のデータ型について，複数の局所変数を用いることに対応する）ことにより，上の条件を満足し，かつ自然なプログラムの記述ができる場合が多い．

6.3 “⊕”，“≺”に関する性質

各$g \in$ DEF(P)の定義文左辺を$g(x_1, \cdots, x_{n_g})$とする．必須頂点列の定義等より，“⊕”，“≺”に関して，次の性質が成り立つ．

（性質6.3） 次の(1)，(2)がともに成り立てば，$u \oplus v$である．

(1) u, vは互いに他の祖先†ではない．

† 以下6.2で親，子供，祖先，子孫というときは，dag表現におけるそれらをさす．

† 6.3で祖先，子孫等というときは，木表現におけるそれらをさす．

(2) u, vの共通の祖先で，レベル（根からその頂点に至る道の長さ）が最大なものをrとし，rの子供でu, vを子孫にもつものをu', v'とすると，次の(2a)または(2b)が成り立つ．

(2a) rのラベルがIFであり，u', v'はrの第2子，第3子または第3子，第2子である．

(2b) label$(r)=f\in$DEF(P)であり，u', v'をrの第i'子，第j'子とすると，V_f内のラベルがx_i', x_j'である任意の2頂点w_i, w_jについて，$w_i \oplus w_j$が成り立つ．

（性質6.4） 次の(1)〜(4)のうち，少なくとも1つが成り立てば，$u \prec v$である．

(1) vがuの真の祖先であり，uを子孫にもつvの子供がvの必須子である．

(2) uがvの真の祖先であり，vを子孫にもつuの子供がuの必須子でない．

(3) u，vは互いに他の祖先ではなく，かつ，u，vの共通の祖先で，レベルが最大なものをrとし，rの子供でu，vを子孫にもつものをu', v'とすると，(3a) u', v'がともにrの必須子であり，$u' \ll_r v'$が成り立つか，または(3b) u'がrの必須子であり，v'が必須子でない．

(4) $u \oplus v$が成り立つ．

7. 最適化の効果

最適化の効果を知るために，本稿で述べた最適化の他に，末端再帰の除去[1]，コンパイル時の書換え[2]を行うASL／Fの最適化コンパイラを試作し[7]†，いくつかの問題を解くプログラムをASL／Fで作成し，目的プログラムの実行効率（実行時間，最大スタック長）を測定した．測定結果については文献(7)を参照されたい．その結果，次のことがわかった．

(i) 必須引数の先評価を行うと，実行時間が2分の1から4分の1に短縮される．

(ii) 必須引数の先評価と末端再帰の除去をともに行うと，最大スタック長が大幅に減少する場合がある．

また，同じアルゴリズムをASL／F，Pascal，Fortranのそれぞれで記述し，その実行時間を測定した[7]．上述の全ての最適化を行えば，ASL／Fプログラムの実行速度はPascalプログラムのそれとほぼ同じであること，Fortranでは不変式のループ外への移動，配列のインデスク計算の工夫，レジスタ割付け等の，本稿では考慮しなかった最適化を行っているので，ほぼ倍程度速いことがわかった．

† 試作コンパイラは，共通部分項の重複計算の除去，ソートの大域化は常に行う．

8. むすび

移植性を考慮して，ASL／Fプログラムを C言語プログラムに翻訳する最適化コンパイラを作成中である．

謝辞 種々の有益な御討論，御助言を頂いた大阪大学基礎工学部杉山裕二博士に深謝する．

文　献

(1) Aho, A. V., Hopcroft, J. E. and Ullman, J. D.: "Data Structures and Algorithms", Addison-Wesley, pp. 66-67 (1983).
(2) Burstall, R. M. and Darlington, J.: "A Translation System for Developing Recursive Programs", JACM, 24, 1, pp. 46-67 (1977).
(3) Henderson, P.: "Functional Programming, Application and Implementation", Prentice-Hall, pp. 214-231 (1980).
(4) Huet, G. and Levy, J.: "Call by Need Computations in Nonambiguous Linear Term Rewriting Systems", INRIA Research Report, No. 359 (1979-08).
(5) 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）．
(6) 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）．
(7) 井上，関，谷口，嵩："関数型言語ASL／Fとその最適化コンパイラ"，信学論(D)，J67-D，4，pp. 458-465（昭59-04）．
(8) 関浩之："関数型言語ASL／Fのコンパイル時における最適化"，大阪大学修士学位論文（1984-02）．

（昭和59年4月4日受付）