C 1　ALGOLコンパイラにおける最適化について

島内剛一（立大理）・広瀬　健（早大理工）
佐久間紘一（京大数理研）・福田康夫（東芝）
志村昭雄（日科技研）

1。はじめに

TOSBAC－3400モデル41のALGOLコンパイラで採用した最適化について報告する．

このコンパイラの言語仕様は，JIS水準7000を基本にしこれに仮パラメータの規制は必ず与えなければならないという制限を加え，またown変数の初期値の設定機能，倍精度実数，複素数，文字等の型を加え，配列代入文を追加するなど若干の拡張をした．

最適化についての基本方針として，まず最適化の処理を他の処理から独立させることにしたが，これは

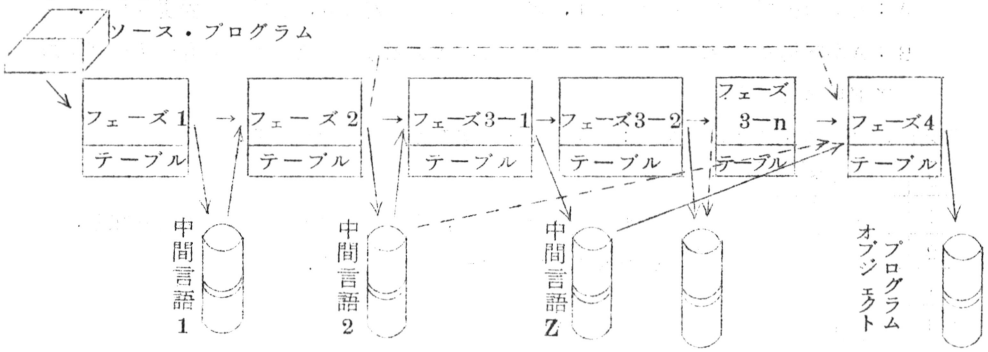
(1) 最適化をオプションにしてプログラマに選択させる．
(2) 最適化を行なう処理プログラムが未完成でもコンパイラのそれ以外の部分が完成すればコンパイラとして使用できる．
(3) 最適化を行なう処理プログラムの機能を自由に拡張できる．

などの利点を生んでいる．

次に，最適化することにより，最適化しないときと結果が異ってしまう（文法で規定しないものも含めて）ことがないよう留意した．

2。コンパイラの構成

コンパイラの構成は第1図に示すとおりである．同図において，点線は最適化を行なわないときの過程を示す．



第1図　コンパイラの構成

まず，フェーズについて説明する．

(1) フェーズ1　構文解析と中間言語1の生成

ソース・プログラムを読み込み，その構文が文法に合っているかどうかを調べ，以後の処理に都合のよいようにテーブルを作成し，中間言語1に変換して作業用ファイルに書き出す．

(2) フェーズ2　中間言語2の生成

フェーズ1で作業用ファイルに書き出した中間言語1を読み込み，文法の制限に触れていないかを調べ，木構造の中間言語2に変換して作業用ファイルに書き出す．

(3) フェーズ3－1～フェーズ3－n

このフェーズは最適化のフェーズで，フェーズ3－1はシステムとして用意されたもので，フェーズ3－2からフェーズ3－nは使用者が必要に応じて作成する．いずれのフェーズも必要のないときは実行されない．各々のフェーズにおいて，木構造の中間言語2を読み込み，同値でより効率のよい木構造に変換し，作業用ファイルに書き出す．

(4) フェーズ4　機械語の生成

フェーズ2またはフェーズ3で書き出した中間言語2を読み込み，機械命令に変換して，作業用ファイルに書き出す．

次に，中間言語について説明する．

(5) 中間言語1

中間言語1はソース・プログラムの各要素を順序を変えずに一定の規則で変換したものである．

変換の規則はつぎのとおりである．

i) 区切りの記号は，その内部コードである2進数に変換する．

ii) 名前，記号列および数は，それを登録したテーブルのアドレスに変換する．

中間言語1の構造は次のとおりである．

A	B

上図で

A：要素の種類で区切り記号は1，名前は2，記号列は3，数は4で表現される．

B：Aが区切り記号のときは内部コードで，Aが記号列または数のときはテーブルのアドレスである．

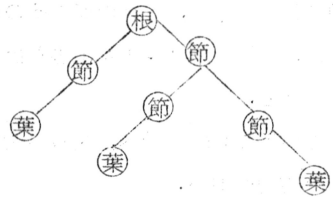
例えば，ソース・プログラム $a:=a+1;$ は次のように変換される．

2	a *
1	$:=$ ↑
2	a *
1	$+$ ↑
4	1 *
1	$;$ ↑

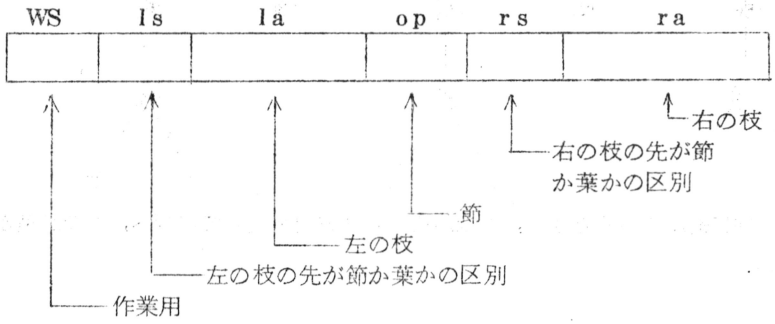
肩に＊のついたものは，そのテーブルのアドレスを意味し，↑のついたものは内部コードを意味する．

(6) 中間言語2

中間言語2は木構造（2進樹木）をした中間言語で，ソース・プログラムのブロックごとに1本の木が作られる．いちばんもとのところを根，末端を葉，つなぎ目を節，節と節をつなぐ部分を枝と呼ぶことにする．

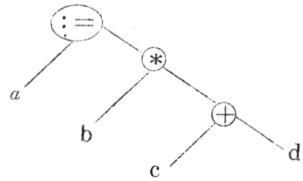


木構造の内部表現は次のとおりである



例えば，a ：＝ｂ＊ｃ＋ｄ；の木構造は第2図のようになるから，内部表現は次のようになる．

	葉	a*	:=↑	節	X_1
X_1)	葉	b*	*↑	節	X_2
X_2)	葉	c*	+↑	節	d*



第2図　a ：＝ｂ＊ｃ＋ｄの木表現

最後にテーブルについて説明する．

(7) テーブル

テーブルには次の4種類がある．

i) block table
ii) identifier table
iii) declaration table
iv) string and number table

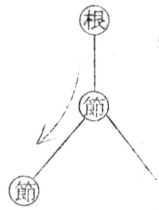
3。 最適化フェーズの詳細

最適化の各フェーズは，ソース・プログラムのブロックごとに作成された木構造の中間言語2を木ごとに読み込み，その木を根から順に節をたどり，各々の節でその節の要素に対応する最適化を行ない，たどり終ると最適化された木を作業用ファイルに書き出すという手順で行なわれる．木のたどり方の規則は次のとおりである．

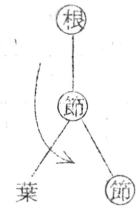
根から葉の方向に進んでいるときは，左枝が節ならば左枝へ，左枝が葉で右枝が節なら右節へ，左右の枝が共に葉なら方向を変えて根の方向へ進む．

すなわち

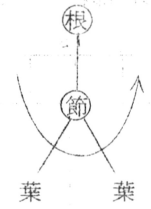
(1)



(2)



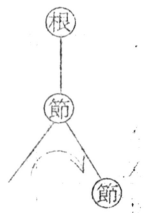
(3)



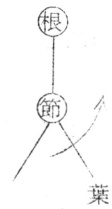
左枝から根の方向に進んでいるときは，右枝が節ならば方向を変えて左枝へ，右枝が葉ならそのまま根へ進む．

すなわち

(4)



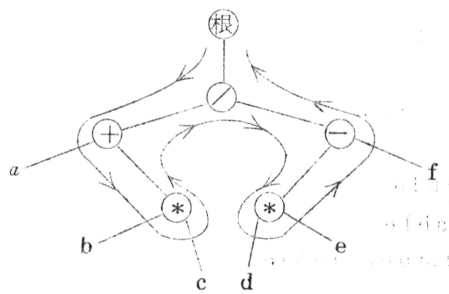
(5)



また右枝から根の方向へ進んでいるときは，そのまま根の方向へ進む．

(6)

式（a+b*c）/（d*e−f）を表現する木について，たどり方を示せば次のようになる．



最適化を行なうのに都合がよいように次のProcedureを用意した．

(1) get tree

```
procedure get tree (f,r);
integer f,r;<procedureの本体>
```

は，fで指定したファイルより木を読み込み，その根のアドレスをrに与えるという機能をもつ．読み込む木がないときは，r＝0とする．

(2) put tree

```
procedure put tree (f,r);
integer f,r; <procedureの本体>
```

は，rを根とする木をfで指定したファイルに書き出すという機能をもつ．

(3) tree trace

```
procedure tree trace (r,n,OP0, OP1 ,……OPn−1,P0,
                                P1 ,……Pn−1);
integer r,n,OP0,OP1,……, OPn−1; procedure P0,P1,……
        ……, Pn−1;
    <procedureの本体>
```

は，rを根とする木の節を順にたどり，その節の要素がOP_0，OP_1，……，OP_{n-1} のいずれかであれば，それに対応する procedure P_i をp_i(r，s)の形で呼び出す．ここで，rは要素の格納されているコアのアドレス，sはたどり方で1から6まで整数である．

これらのprocedureを使って，最適化フェーズは次のように書くことができる．

```
begin integer u,v,r;
    procedure p0(r,s);
      integer r,s;<procedureの本体>
    procedure p1(r,s);
      integer r,s;<procedureの本体>
        ⋮
    procedure Pn−1(r,s);
      integer r,s;<procedureの本体>
        u:=α;  v:=β;
  L:get tree (u,r);
        if r=0  then go to OWARI;
        tree trace (r,n,OP0,OP1,……OPn−1,P0,P1,…Pn−1);
        put tree (V,r);
        go to L;
  OWARI:end
```

上のプログラムでα，βは，各々入力と出力のファイルに対応する整数であり，procedure pi は木を改良するprocedureで，改良するための条件を満たしているかどうかを調べて，満たしていれば改良を行ない，そのアドレスをrに与える．

次に，整数巾乗を整数回の掛算に変えるprocedureをあげておく．

```
procedure PTOM (s,r);
  integer s,r,W1,W2,i;
begin if tree rs(r)=1 ∧ kind (tree ra(r))=8
      ∧type (tree ra(r))=1∧in value(tree ra(r))≦7
      ∧in value(tree ra(r))≧2 then
  begin W1 := r; W2 :=tree ra(r);
    generate tree (r,0,tree ls(W1), tree la(W1)
    ,*↑, tree ls(W1), tree la(W1);
    W2 :=W2−2;
    for i :=1 step1 until W2 do
      generate tree(r,0,2,r,*↑, tree ls(W1)
      ,tree la(W1));
  end
end PTOM;
```

上のプログラムで使われているprocedure は各々次の機能をもつ．

(1) kind

パラメータで指定した量の種類(Simple variable, array, label, Switch, procedure, String, numberを値としてもつinteger procedureである．

(2) type

パラメータで指定した変数または数の型を値としてもつinteger procedureで，型の値は整数型は1，実数型は2などである．

(3) i n value

パラメータで指定した整数型の数の値を値としてもつinteger procedureである．

(4) tree ls , tree la , tree rs , tree ls

パラメータで指定した木要素のls，la，rs，ra，の部分の値を値としてもつinteger procedure である．

(5) generate tree

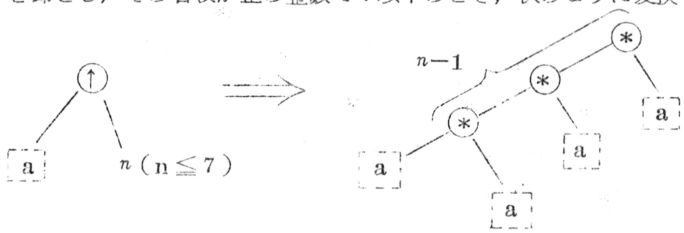
第2のパラメータ以降で指定した値をもつ木要素を作りそのアドレスを第1パラメータの変数に与えるprocedureである．

4。 最適化の例

$a x^4 + b x^3 + c x^2 + d x + e$を最適化する手順について説明する．

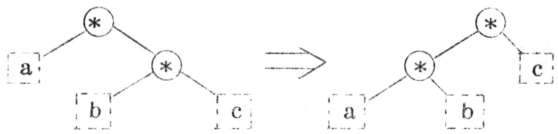
これに使われる最適化の基本操作は次の3操作である．

(1) 演算子↑を節とし，その右枝が正の整数で7以下のとき，次のように変換する．



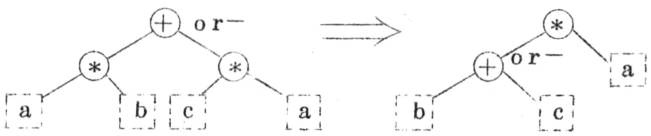
巾乗は普通サブルーチンで計算するので演算時間が多くかかる．演算時間の少なくてすむ乗算に変えることは有効である．

(2) 演算子＊を節とし，右枝も演算子＊をもつ節になっているとき，次のように変換する．

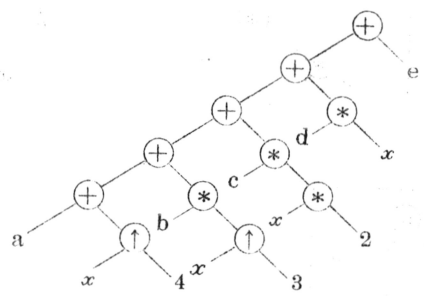


右下りの木は中間結果の退避，回復が必要である．退避，回復の必要のない左下りの木に変えることは有効である．

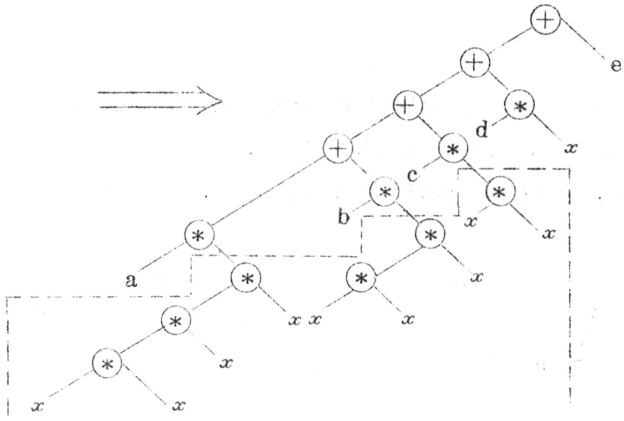
(3) 演算子＋または－を節とし，その左右の枝が共に演算子＊をもつ節で，共通な枝を持つとき，次のように変換する．



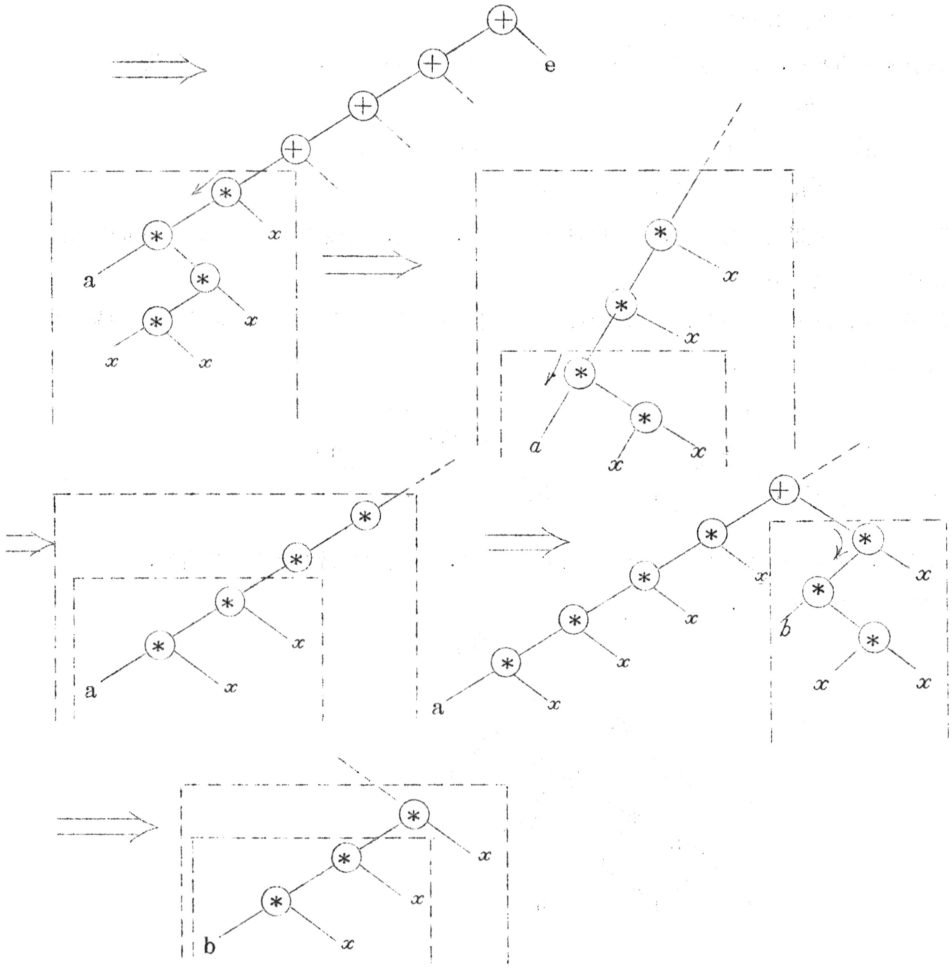
ところで，$a x^4 + b x^3 + c x^2 + d x + e$は，最適化を行なわない場合は，次のような木構造で表現される．



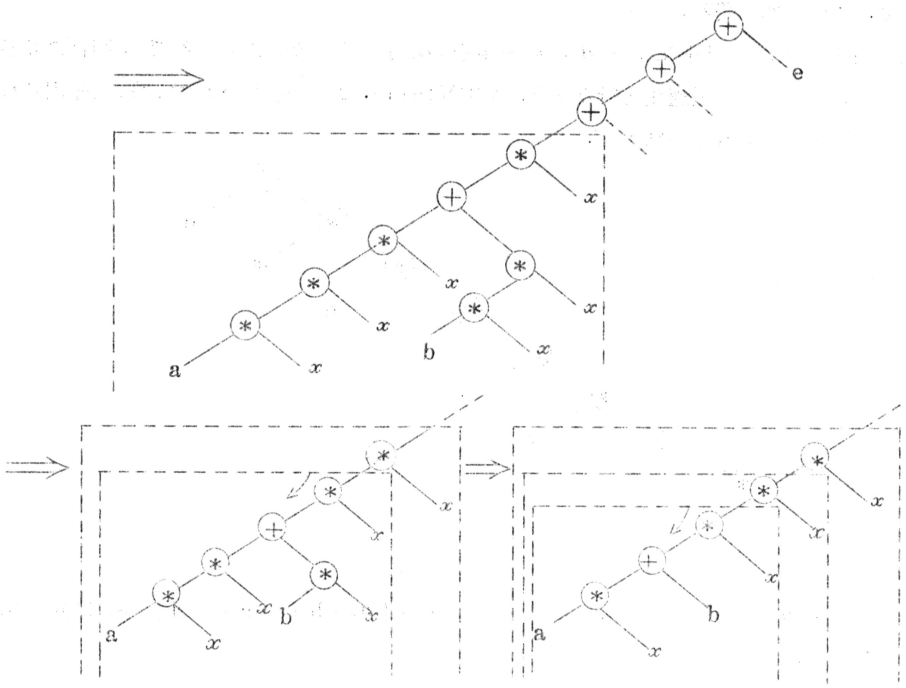
上記の基本操作(1)によって最適化を行なえば



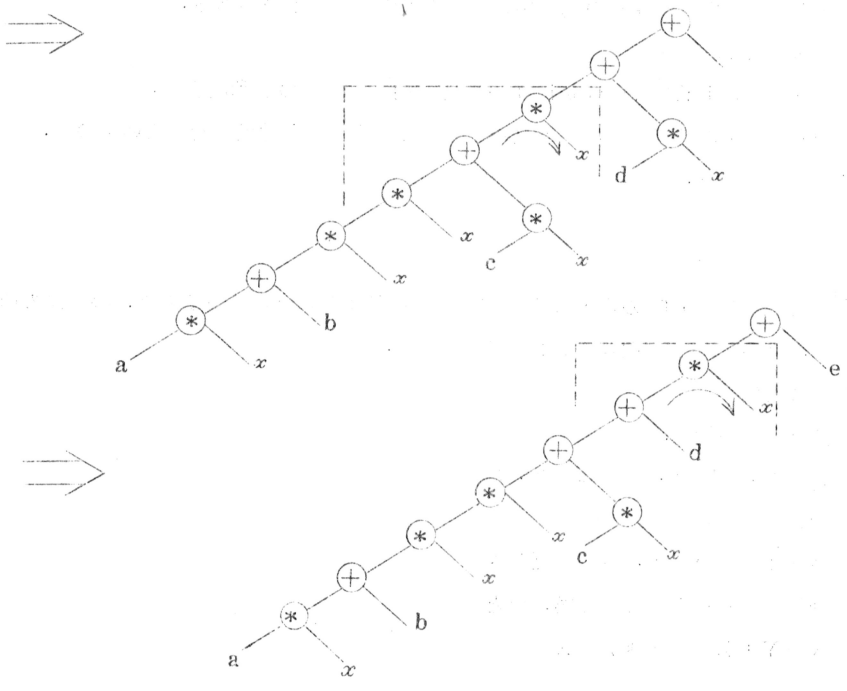
さらに，最適化基本操作(2)により



となり，次に最適化基本操作(3)により

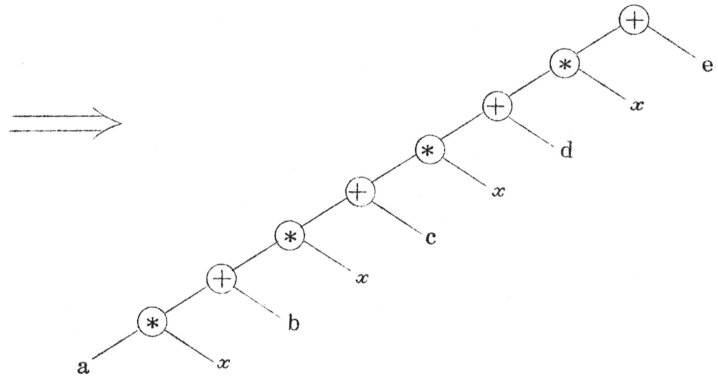


さらに



となり，この操作を終る．

結局（（a x＋ b） x^2＋ c x ＋ d）x ＋ eを表現する木になったところで 1回目の最適化が終り，ここで再び最適化を行なうと，基本操作(1),(2)は適用されず(3)が適用され，次のような木が作られて完結する．



すなわち，a x^4 ＋ b x^3 ＋ c x^2 ＋ d x ＋ eが（（（ax＋b）x＋c）x＋d）x＋eに変換されたわけである．

5. 最適化の基本操作の種類

最適化の基本操作には4章の例で用いたもの以外に次のものがある．

(1) 繰り返し文の最適化

i) 繰り返しにindex registerを利用できるものは利用する．

ii) 繰り返し文の中にあり，繰り返し文の外で計算しても効果の変らないものは，外で計算するようにする．

など

(2) 条件文の最適化

条件文においてif のあとの論理式が比較式だけからなっているとき，論理式の値を計算せずに，直接比較式を判定するように変換する．

(3) 式の最適化

i) 数と数との計算，たとえば

X＋1．0／2．0 ⇒ X＋0．5

ii) 数の型変換，たとえば

X＋1 ⇒ X＋1．0　ただしXは実数型

iii) 演算子の左辺と右辺の変換，たとえば

X＋Y＊Z ⇒ Y＊Z＋X

6。サイド。エフェクト

最適化を行なうことにより，一次子の評価の順序が変ることがしばしばおこるが，関数を含んだ式では評価の順序を変えると結果が異なることがある，

たとえば

```
begin integer A,Y;
 integer procedure P(F);
       integer F;
       begin P := F;
             F := F+1
       end;
      A := 2;
      Y := A+P(A) * 2
eud
```

上記プログラムでは式A＋P（A）＊２を左から順に評価し演算すると２＋２×２で ６となるか，同じ式をP（A）＊２＋Aと変換し左から順に評価し演算すると２×２＋３で７になり結果が異なる．

ＪＩＳ規格では一次子の評価の順序を変えることにより結果の異なる式の効果については規定していないが，ＩＳＯ規格では左から順に評価するようにと規定している．

我々のコンパイラでは，ＩＳＯの規格に従い関数を含む式であるかどうかを調べ関数を含む式であれば，その式の最適化はあきらめることにした．

7。最適化基本操作の適用の順序

最適化基本操作の設定およびその適用の順序は重要である．不適当に適用すると，適用する回数が多くなるばかりでなく，悪い木に変えてしまうことになる．改悪の危険性については，基本操作を吟味して選ぶことによって避けることができるが，適用の順序は，ソース・プログラムの性質によって異なる．たとえば昇巾の順の多項式と降巾の順の多項式では最適化の基本操作の適用の最適な順序は違ってくる．現在は何通りかの順序を用意し，ユーザにソースプログラムの特徴により選ばせるように考えている．

8。おわりに

中間言語をリスト構造で表現することは，ＩＢＭ３６０ＦＯＲＴＲＡＮでも利用されており，特に新しい試みではない．しかし最適化には都合がよく，その構造も単純リスト，二進樹木などごく単純なもので十分である．最適化を行なう場合に注意しなければならない問題は，第１章にもあげたが，最適化による副作用である．ＡＬＧＯＬはもともと論理的な正確さを重視するとい

うことで，FORTRANのように処理効率などを重視しなかったという歴史的な背景があり，一次子の評価順序にしても左から順に評価されると信じられているようである（この点についてはJIS規格では規定していない）．我々のコンパイラでは，この迷信を重んじ，評価順序を変えると結果が変る可能性があるものは順序を変えないという方法をとったが，そのためにコンパイラの処理効率をかなり下げることになってしまった．ALGOLを実用にたえる言語にするためにはこれら迷信を打破っていく必要があるし，さらにコンパイラがより効率のよい目的プログラムを作り易いよう言語仕様を変える必要があろう．