ポインタ型データを含む言語の共通式検出による最適化

7B－7

清野ひろし　岩澤京子
東京農工大学工学部電子情報工学科

１．はじめに

Ｃ言語によるソフトウェア資産は多い．Ｃ言語では，ポインタ抜きのコーディングが難しい．

プログラミングの生産性，信頼性の向上のためには，モジュール化，カプセル化を図ることが重要である．Ｃ言語でも小規模な関数を組み合わせてプログラムを作る．複数の値を関数で求める場合もポインタ使用は不可欠である．

しかし，最適化のためにはポインタが何を指しているかを調べる必要がある．ポインタが指す対象を絞り込むことは，プログラムの規模が大きくなればなるほどメモリサイズと処理時間が増加するため難しくなる．本研究では，ポインタ変数の解析とその解析結果を利用した不要なコードの削除を目的とする．

２．本システムの設計方針と概要

本システムは中間語に対して，最適化を施す．中間語は比較的，機械と独立のため本研究は多くの言語に適用可能である．入力した中間語の語数を，１～２割程度減らした中間語を出力することを目指す．

基本ブロックに対してＤＡＧを生成することで共通式を認識する．このとき，ポインタによる間接代入があれば，ポインタが何を指しているかを調べる．解析対象の識別子の使用と定義をたどる．解析情報は中間語数ｎ，識別子数ｍ，基本ブロック数ｂに対して，ｎ×ｍ×ｂに比例して増大する．そのため外部メモリに解析情報を持たせる．また，関数単位に中間語を管理することにする．

処理対象のポインタは，ローカルなものとする．基本ブロック内の最適化であるため，主な処理対象は一時的なポインタ変数と関数内部のローカルな変数だからである．グローバルなポインタへの間接代入が出現したら，当該基本ブロックではすべての識別子を共通式認識からはずす．図１は機能概要図である．

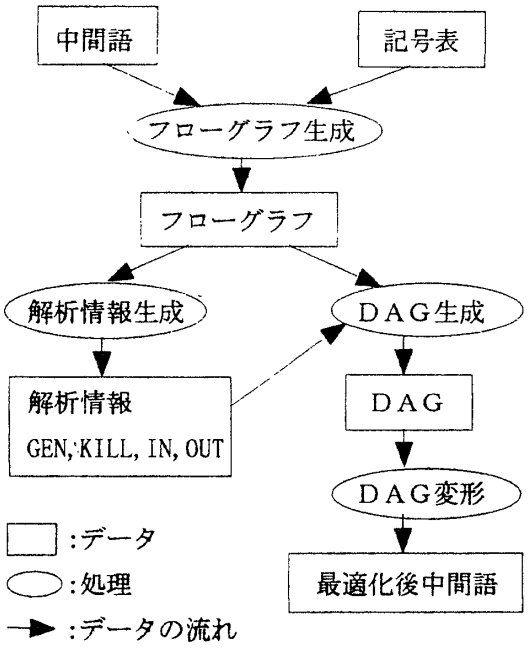

図１　機能概要図

アドレス演算子と間接代入の組合せによって，ＤＡＧが簡略化できることも示す．

３．本システムの実現方式

本システムのデータ構造は，関数単位のフローグラフによって管理されている．（図２）　動的にリンクされた情報も含めてファイルにデータを保存し，大規模な解析を可能にしている．動的にリンクされたアドレスはメモリブロック単位に管理され，ベースアドレス表へのポインタとオフセットにより表現されている．ブロック単位に転送することで，メモリとファイルとの間のロードセーブの手間を減らすようにした．

Optimization of Detecting Common Subexpression for Pointer Data
Hiroshi Kiyono, Kyoko Iwasawa
Tokyo University of Agriculture and Technology

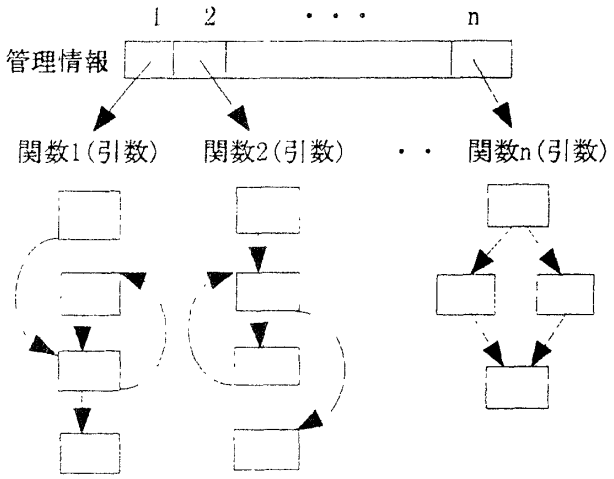

図２　フローグラフの管理構造

中間語は四つ組とし，共通式の認識はＤＡＧ生成で行った．ＤＡＧの定義や生成方法は，文献[１]に従った．さらに本システムでは，ＤＡＧによってすべての基本ブロックを表現させるために，関数呼出やポインタに関して次のような拡張を行った．表のfoo(1)は関数fooへの2番目の引数であることを示す．

表１　拡張したＤＡＧ表現

	間接代入	間接参照	アドレス	引数
中間語	*p:=x	x:=*p	p:=&a	PAR a, foo(1)
ＤＡＧ表現	p → L* → x	x → R* → p	p → & → a	foo(1) → P= → a

目標とするＤＡＧの簡略化例を挙げる．次のような基本ブロックに対して，ＤＡＧの簡略化を行う．識別子T01, T02は一時変数とし，このブロック以外では使用されないものとする．

```
T01 := &b
a := b + c
T02 := *T01
d := T01 + c
```

この基本ブロックからＤＡＧを生成すると，図３(a)のようになる．葉から"&"と"R*"でつながっているので，代入":="と置き換え，"T01"を削除すると図３(b)になる．"b"と"T02"が代入でつながる．さらに"T02"を削除すると"d"の左の子が"b"になる．ここで"b+c"が共通式となるので，図３(c)のように簡略化された．中間語を再構成すると(d)になり，命令数の削減と低コストの代入への変換ができた．

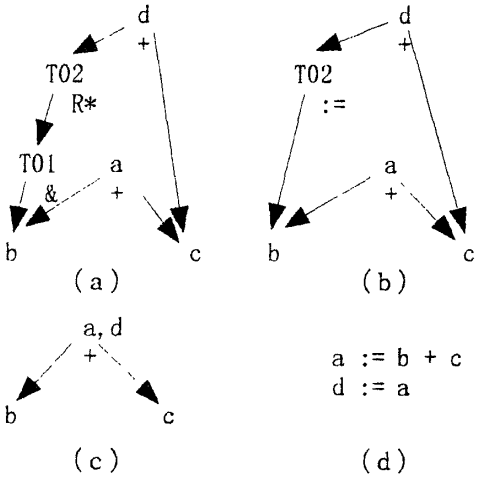

図３　ＤＡＧの簡略化

４．終わりに

本システムの実行結果について考察する．入力に使用した中間語の生成は，フリーソフトのＣサブセットコンパイラに中間語生成機能を付加し，行った．入力した中間語は，最小自乗法の配列による計算部分である．元のＣのソースは60行程度である．中間語数は410語であった．出力された中間語数は354語に減少した．減少した56語のうちアドレスのセットは6語，ポインタ加減算が7語，である．一時変数は，入力時253個あったが，出力時には199個となった．減少した54語のうちポインタとして使われたものは，15個である．入力時のポインタ型一時変数は68個である．

減少した中間語数は入力語数の14%である．しかし，ポインタに関してはあまり効果的ではなかった．これは，現システムの最適化が基本ブロック内だけであるためと考えられる．また一時変数全体としては21%減少し，そのうちポインタ型だけで見ると22%減少した．四つ組は一時変数の個数が多くなりやすいが，ポインタ型一時変数を減らす効果はでていると考えられる．

基本ブロックをまたがった解析や別名の解析が今後の課題である．

参考文献[1]エイホ，ウルマン コンパイラ 倍風館 1986