プログラミング－言語・基礎・実践－ 20－3
（1995. 1. 27）

ϕ- 関数移動による効率的な部分的冗長計算除去

滝本 宗宏　　原田 賢一

慶應義塾大学大学院 理工学研究科 計算機科学専攻

本稿では，二次的効果を含む部分的冗長計算除去の効率的な手法を提案する．本手法は，過去の研究において，同時におこなわれていた等価計算の発見と計算点の決定を分離し，さらに，等価計算の発見をフローグラフとは，独立に行う特徴をもつ．また，本研究において，ϕ- 関数移動と静質問伝播という２つの新しい考えを導入する．ϕ- 関数移動は，異なった場所に定義の結合をもつ式の等価性を露出し，また，制御の流れに依存して等価となる計算式を，フローグラフと独立に見つけ出すことを可能にする．静質問伝播は，付加的代入の導入過多を防ぐために提案された，質問伝播を変形したものであり，データフロー方程式にもとづく効率的な解析を可能とする．

Efficient Partial Redundancy Elimination Based On ϕ-Function Motion

Munehiro TAKIMOTO　　Kenichi HARADA

Department of Computer Science, KEIO UNIVERSITY

This paper presents an efficient and effective algorithm for partial redundancy elimination. This paper proposes separation of exposing new opportunities to detect new redundancies and determining computation points to insert terms ,which makes the algorithm efficient and simple. As part of our work, we introduce new algorithms of ϕ-function motion and static question propagation. The ϕ-function motion enables independently of flow graph to expose equivalency between computations which have different join locations of definitions and to detect equivalency of computations which depends on program paths. The static question propagation enable to suppress introducing too many additional assigns efficiently.

1 はじめに

冗長計算除去の手法は、コンパイラのコード最適化において非常に強力な手法であり、以前から多くの研究がなされてきた。その大半は、共通部分式を、除去の対象としている。ここで、共通部分式とは、式 E が現われ、それ以前にも同じ形の式があり、それらの間で、E で使用している変数の値が変更されないような計算式 E のことである [ASU]. 一つの共通部分式の除去は、新たな共通部分式を副次的に生む可能性があるため、プログラム全体にわたって、複数回の除去が必要である.

この二次的効果を効率良く利用する手法として、大域値番号付け (Global Value Numbering)[RWZ] がある. 低いランクから順に、ランクの同じ計算式に対して、コード巻き上げ (Code Hoisting) による共通部分式の除去を行う. フローグラフに沿って、コード巻き上げを行うために、各基本ブロック、各辺に対して、式を管理する表を必要とし、プログラムは構造化されている必要がある.

共通部分式の除去により新たな共通部分式を生み出す過程は、式の等価性を見つけ出す過程と言える. この考えをもとにしたものに、値流れグラフによる手法 [SKR1][SKR2] がある. エルブラン解釈にもとづいて、同じ値を計算する式を辺でつないで値流れグラフ (Value Flow Graph) を作る. この値流れグラフ上で、データフロー解析によるコード巻き上げ [MR] を行って、冗長計算を除去する. この手法は、フローグラフに沿って、各節におけるエルブラン解釈を計算するため、大きなコストを必要とする.

本研究の手法も、値流れグラフによる手法同様に、等価性の認識と、冗長計算の除去の 2 段階で最適化を行うが、フローグラフとは独立した、式の依存グラフである値グラフ (Value Graph) にもとづいて、式の等価性を導き出す. ただし、他の値グラフにもとづいた等価性認識手法 [AWZ][RL1][RL2] と異なり、結合 (Join) 性の違いから生ずる式の表面上の違いによらず、式の等価性を認識し、かつ、制御の流れに依存して、等価になり得る式も見つけ出す必要がある. このため、新たに、ϕ- 関数移動という考えを導入する. ϕ- 関数移動は、式の意味を変えることなしに、値グラフ上で、ϕ- 関数にあたる ϕ- 節を移動することによって、定義結合の違いにより隠された式の等価性を露出しようという考えである. この手法は、二次的等価性認識と、本質的に同じ考えにもとづいている.

この後、次節では、計算式の表現として利用する値グラフについて述べ、等価性を発見する際の ϕ- 関数の問題点を述べる. 3節では、2節で述べた ϕ- 関数によって発見できない等価性の露出法として、ϕ- 関数移動という手法を導入する. 4節では、ϕ- 関数移動を利用した等価性発見アルゴリズムを示す. また、ϕ- 関数移動のための値グラフの拡張について述べる. 5節では、ϕ- 関数移動で得られた値グラフを用いて、どのように、値流れグラフを構築するか述べる. 6節においては、一時変数の導入過多への解決として、静質問伝播 (Static Question Propagation) を提案し、大域的値番号付け手法の質問伝播 (Question Propagation) と比較を行う. 7節において、本研究の目標である最適化のための、計算点と冗長計算の決定のしかたを示す. 最後の節において、まとめとして、本研究の有用性と、これからの研究について述べる.

2 値グラフと結合性の問題

2.1 値グラフ

値グラフ [RL1][RL2][AWZ] は、ラベルの付いた有向グラフである. 辺は、変数の使用と、その値を生成する定義の結び付きを表している. 定義の中には、$A \leftarrow B$ のように、ある変数の値を別の変数が受け取るだけのものが存在する. この場合は A の値の生成は B の定義でなされるとする.

値グラフの節は、プログラム中の個々の関数に一致している. 節は、定義の種類によって、2 種類の形式をもつ.

実行可能関数 定義の右辺上の関数記号でラベル付けした節で、関数の各引数に対して、一つの辺が出ている.

ϕ- 関数 同じ変数による値の結合を表すもので、ϕ という仮想関数記号によってラベル付けした節である. 結合する各定義に対して、一つの辺が出ている.

値グラフは、SSA 形式 (Static Single Assignment Form) に変形したプログラムから、容易に作ることができ、ϕ- 関数は、この SSA 形式で導入される仮想関数である. SSA 形式は、各変数の使用に対して、その定義が唯一となるように変形したプログラム形式のことである. 異る定義が結合する部分には、新たに ϕ- 関数による定義を導入することによって、使用に対して唯一の定義という関係を表現する. ϕ- 関数は $U \leftarrow \phi(V, W, \cdots)$ の形をしていて、$U, V, W, \cdots$ は変数である. 引数 $V, W, \cdots$

は ϕ- 関数が置かれているプログラム点の先行節の数と一致している．j 番目の引数は，j 番目のフローグラフにおける先行節と結び付いてる．ϕ- 関数に j 番目の先行節から制御が達すると，U には j 番目の引数の値が代入される．ここで重要なことは，ϕ- 関数の実行は引数の一つだけを使用するということである．

図 1はプログラムを SSA 形式に変換した様子を示している．図 2はその値グラフである．

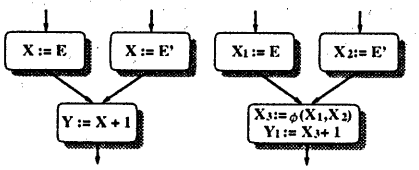

図 1: SSA 形式

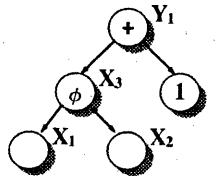

図 2: 値グラフ

2.2 結合性

値グラフは，式の構造を大域的に表現するという意味で優れた表現形式であるが，定義の結合を表すために，明示的に ϕ- 関数を導入するので，その出現位置の相違が，値グラフの構造の違いとなり，本来，等価である計算式を発見できないという状況が生じる．

図 3の P と Q は同じ値を計算する．ここで，P は最初の加算が行われる前に，変数 X によって定義が結合しているのにたいし，Q は最初の加算の後に変数 Y によって結合している．この違いは，ϕ- 関数による値グラフの構造の違いとなって現われる．

次の節では，この結合性の違いによって隠された等価性を露出する手法として，新たに，ϕ- 関数移動という考えを導入する．

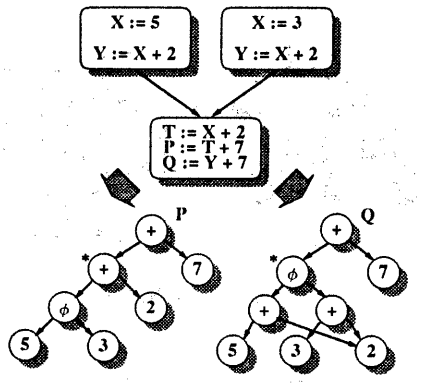

図 3: 結合の違いによる異なった値グラフ表現

3 ϕ- 関数移動

前節で述べた ϕ- 関数の出現場所の相違の問題を解決するために，意味を変えずに，値グラフを変形することを考える．変形は，ϕ- 関数に着目して，次の 2 種類が考えられる．

- ϕ- 関数降下
- ϕ- 関数巻き上げ

この節では，それぞれの ϕ- 関数移動について述べ，本研究で，実践的にどのように ϕ- 関数移動を導入するかを述べる．

3.1 ϕ- 関数降下

ϕ- 関数降下を次のように定義する．

定義 3.1 (ϕ- 関数降下) 値グラフ VG 上に，ある ϕ- 節 V_ϕ が存在して，n 個の各節 $V_i (for\ i = 1, 2, \cdots, n)$ に依存しているとする．すべての i について，V_i のラベル (ϕ- 関数ではない) は同じであり，各 V_i は m 個の依存先 $V_{ij} (for\ j = 1, 2, \cdots, m)$ をもつとする．

V_i と同じラベルをもつ節 V' を生成して，V_ϕ と置き換える．各 V_i の j 番目の依存先を i 番目の依存先とする ϕ- 節 (V_ϕ と同じラベル) を，すべての j について生成し，それらを $V_{\phi j}$ とする．各 j に対して，V' の j 番目の依存先を $V_{\phi j}$ とする．このとき，$V_{\phi j}$ の中で，その依存先がすべて同じ節であるものは，$V_{\phi j}$ の依存先の節を V' の依存先とする ∎

ϕ- 関数降下の様子を図 4に示す．

この定義のもとで，次の定理が成り立つ．

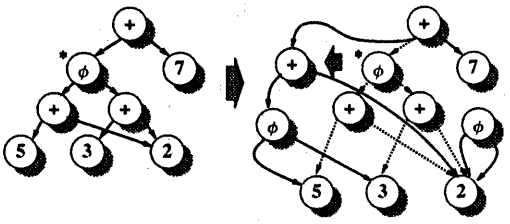

図 4: ϕ- 関数降下

定理 3.1 ϕ- 節 V_ϕ は, ϕ- 関数降下の適用によって, 意味を変えない.

証明. 今, ϕ- 関数が置かれている基本ブロックの n 個の先行節のうち k 番目から制御が移ると仮定する. このとき, V_ϕ の値は, V_ϕ の k 番目の依存先 V_k を根とするサブグラフの値に等しい. また, V_k は依存先として, $V_{kj}(j$ は V_k の表す関数の j 番目の引き数であることを示す) をもつ.

一方, ϕ- 関数降下の適用によって, V_ϕ と置き換えられる V' は, V_i と同じラベルをもっている. V' の j 番目の依存先が ϕ 関数 $V_{\phi j}$ であるとき, そのラベルは V_ϕ と同じものであり, その値は, $V_{\phi j}$ の i 番目の依存先 $\bar{V}$ を根とするサブグラフの値に等しい. 定義により, $\bar{V}$ は V_{kj} であるから, V' と V_k は同じラベルをもち, その引き数にあたる依存先が等しい. よって V' と V_k は等価であり, V' と V_ϕ は等価である. ∎

定理 3.1より, ϕ- 節を根とするサブグラフの表す式の意味は ϕ- 関数降下の適用によって変化しない. よって, その ϕ- 節を子孫としてもつ節の意味も変えない.

3.2 ϕ- 関数巻き上げ

ϕ- 関数巻き上げを次のように定義する.

定義 3.2 (ϕ- 関数巻き上げ) 値グラフ VG 上に, ある ϕ- 節でない節 V が存在して, m 個の各節 $V_j(for\ j = 1, 2, \cdots, m)$ に依存しているとする. V_j の中には ϕ- 節が含まれており, かつその種類は 1 種類とする. 特にこの ϕ- 節を $V_{\phi j}$ とし, その n 個の依存先を $V_{ji}(for\ i = 1, 2, \cdots, n)$ とする.

ラベルが V と等しい n 個の節 V'_i を生成し, それらを依存先とする ϕ- 節 V_ϕ を V と置き換える. V に対応する V'_i の m 個の依存先のうち, V において, ϕ- 節でない依存先は共通の依存先とする. ϕ- 節である依存先 $V_{\phi j}$ については, 各 $V_{\phi j}$ に対して, i 番目に対応する依存先を V'_i の対応する j 番目の依存先とする. ∎

ϕ- 関数巻き上げの様子を図 5に示す.

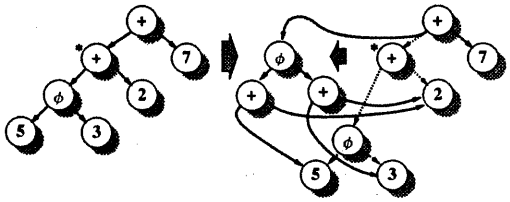

図 5: ϕ- 関数巻き上げ

ここで, V の依存先は, せいぜい 1 種類の ϕ- 節としたが, 実際は異なったものが存在して良い. しかし, ϕ- 関数には実行順序があるため, 1 回の ϕ- 関数巻き上げで考慮に入れるのは 1 種類の ϕ- 節である. つまり, 処理の対象でない ϕ- 節は, ϕ- 関数をラベルにもたない節と同様に扱えばよい. ここで ϕ- 節の処理順序ということが問題になる. 一般的に, ϕ- 節の処理順序が決定できない場合が存在する (フローグラフが構造化されていないとき). この処理順序の戦略については, 4節で述べる.

定理 3.1同様次の定理が成り立つ.

定理 3.2 ϕ- 節でない節 V は, ϕ- 関数巻き上げの適用によって, 意味を変えない.

証明. 定理 3.1と同様. ∎

定理 3.2より, ϕ- 節でない節を根とするサブグラフの表す式の意味は ϕ- 関数巻き上げによって変化しない. よって, その節を子孫としてもつ節の意味も変えない.

図 3において, Q の ϕ- 節 (* の付いた節) に ϕ- 関数降下を適用すると, P のグラフとなり, P の節 (* の付いた節) に ϕ- 関数巻き上げを適用すると Q のグラフとなる. ϕ- 関数降下と ϕ- 関数巻き上げは互いに逆写像の関係にある.

3.3 等価性露出への応用

ϕ- 関数移動を適切な順序で反復適用することによって, 異なった, 定義の結合をもつ計算式の等価性を露出することができる. ただし, 多くの冗長計算を発見するということを考えるとき, 部分的

冗長 (Partial Redundancy) を発見することが必要になる．部分的冗長とは，ある制御の流れに限って，冗長となるものである．等価な式は，すべての制御の流れに対して，同じ値が計算されることを前提としている．ϕ- 関数降下はこの前提のもとに ϕ- 関数を降下させるため，制御の流れに依存して，同じ値を計算する式を保持することはできない．これにたいし，ϕ- 関数巻き上げは，ϕ- 節でない節を ϕ- 節にしていくため，ϕ- 節となった節は，ϕ- 節の依存辺を介して，制御に依存して同じ値を計算する節を保持する性質をもつ．よって，次節では，冗長計算除去を目的とする，等価性発見アルゴリズムとして，ϕ- 関数巻き上げを用いた手法を述べる．

4 ϕ- 関数巻き上げによる等価性発見

この節では，値グラフが構築されていることが前提となる．また，値グラフは，再帰変数の等価性発見のため，[AWZ] の分割 (Partitioning) アルゴリズムによって，前段階の等価性発見が行われており，一致している節は一つに合体されているものとする．これは，[RWZ] において，SSA 形式を還元 SSA 形式に変換しておくのと同様である．

4.1 値グラフの拡張

値グラフに対して，ϕ- 関数巻き上げを適用すると，実行可能関数 (ϕ- 関数でない関数) が，ϕ- 関数に置き換えられる場合がある．ここで，節の置き換えという形で表現したが，節はそのままで，ラベルと依存先が変わる過程だともいえる．そこで，本研究で使用する値グラフは，次の点で拡張を行う．

- 拡張値グラフの節は，値グラフの節を，2 種類保持することができる．その値グラフの節の一方は ϕ- 節であり，もう一方は ϕ- 節でない節である．
- 拡張値グラフの辺は，値グラフの辺と一致し，各節からでている辺は各拡張値グラフの節が保持している値グラフの節からでている辺である．辺の先は拡張値グラフの節である．

初期の拡張値グラフは，値グラフから容易に構築することができる．各値グラフの節を，拡張値グラフが保持する一つの節として，登録する．そして，値グラフの辺の指している先を値グラフの節から，その値グラフの節を含んでいる拡張値グラフの節に変更する．

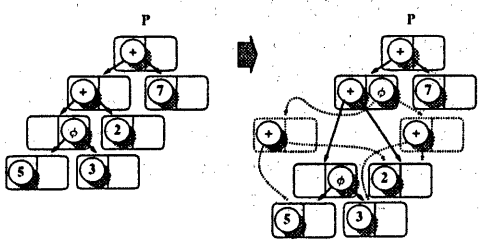

図 6: 拡張値グラフと ϕ- 関数巻き上げ

初期の状態では，各拡張値グラフの節は，それぞれ一つだけの値グラフ節だけを保持している．

図 6において，初期状態の拡張値グラフと，ϕ- 関数巻き上げ適用によって，ϕ- 節の欄が埋まった様子を示す．

4.2 ランク付け

拡張値グラフには，依存先から評価が行えるように，評価順序を付ける．これをランクという．

まず，フローグラフにおいて，各基本ブロックに，深さ優先の逆順の順序 (トポロジカルソート順序) 付けを行う．値グラフの各節に，その節に相当する式が存在している基本ブロックの順序を記録しておく．

この後，次の規則に沿って，計算する．

- 変数が，プログラムの入り口名 V_0 だった場合，ランクは 0 である．
- 代入される変数のランクは，代入される式のランクである．
- 式が定数だった場合は，ランクは 0 である．
- 式が ϕ- 関数だった場合，式のランクはオペランドの最大のランクである．
- 式が，変数，定数，ϕ- 関数のどれでもない場合，式のランクはオペランドの最大ランクに 1 を加えたものである．

ランクが 0 の値グラフ節から順番にランクを計算し，割り当てていく．このランクの計算法は [RWZ] と同じである．ループヘッダの式のランクを計算する際に，バックエッジに沿って，ループヘッダに到達する変数のランクが必要になる場合があるが，その変数のランクの値は，0 とすることによって計算していく．この変数は値グラフ上にトポロジカ

ルソート順序が記録してあるため，その順序の逆転によって，特定することができる．

本研究における等価性発見アルゴリズムの1ステップは，ランクの小さい順に，1度づつ実行するようになっている．このことは，どんな計算も，その引数を生成する計算がすべて処理されてから，処理されることを保証する．

4.3 操作

拡張値グラフ上の各節のうち，同じ値を計算している節を合体させていく．この操作は各ランクに対して，1回，ランクの低い順に行う．操作全体において，等価計算発見のために，計算を保持するハッシュ表 GCT(Global Computation Table) を用意する．拡張値グラフ節に含まれる，各値グラフ節は，対応する GCT エントリへの参照をもっている．この参照は，等価な節を合体させた際の，登録更新に用いる．また，等価計算を発見するために，関数名と，引数にあたる拡張値グラフ節の組によって，エントリを検索することができる．GCT は頻繁な，計算の登録，削除を容易にし，効率をあげる役割をする．大域値番号付けにおいても，計算を保持するために，各基本ブロックと各フローグラフ辺にハッシュ表を用意しているが，本研究では，フローグラフ上での計算の移動を行わないため，表はプログラム全体で一つとなる．

ランク R の拡張値グラフ節 V_R が依存している，または，V_R から ϕ- 節だけを通して到達できる，ϕ- 節のセットを V_ϕ とする．各ランク R で行う操作を次のように定義する．

- V_R 内の各節に対して，GCT を検索し，同じ計算をしている節 と置き換えることによって，節の合体を行う．
- V_ϕ 内の ϕ- 節について，逆トポロジカルソート順序で，V_R 内のすべての節に対して，ϕ- 関数巻き上げを行う．この際，ϕ- 関数巻き上を適用するごとに新しくできる節に対して，GCT を検索し，同じ計算をしている節があれば，その節を新しい節で置き換える．
- ϕ- 関数巻き上げの適用によって，ϕ- 関数をもつようになった節 (元から ϕ- 節だけで，変化しなかったものも含む) を，ϕ- 節と見て (拡張値グラフの節は，ϕ- 節と通常の関数の2種類の意味をもつ)，トポロジカルソート順序で，同じ計算をしている節を見付け，一つの節にしていく (片方の節で置き換える)．このとき，すべての依存先が同じ ϕ- 節の消去も同時に行う．

各ランクにおける ϕ- 関数巻き上げの適用が多くなると，ϕ- 関数巻き上げによって新たに生成される節によって，拡張値グラフのサイズは指数的に大きくなる．そこで，実践的な，ϕ- 関数巻き上げの制限を導入する．それは次のようなものである．

- ϕ- 関数巻き上げの適用により新しく生成された節のうち，過去において，同じ ϕ- 関数をもつ ϕ- 節による ϕ- 関数巻き上げの適用があった場合，その ϕ- 関数をラベルにもつ ϕ- 節による適用は行わない．

この制限の導入によって，本研究の等価性発見アルゴリズムは，悲観的にみて，計算量 CJ で抑えられる．ここで，C は，プログラム中の計算式の数であり，J はフローグラフ上で制御の結合となっている基本ブロックの数である．

実装における制限の実現は次のようにする．拡張値グラフの各節に，派生元として，ϕ- 関数巻き上げ適用前の節を保持させ，ϕ- 関数巻き上げで新しく節ができるたびに，その節に派生元の節を伝播する．また，各 ϕ- 関数ごとに，ϕ- 関数巻き上げを適用した節を保持しておくようにする．

ϕ- 関数巻き上げを適用する前に，適用しようとしている節の派生元が，巻き上げに使用される ϕ- 関数の適用済み節に含まれていないか確かめる．もし，含まれているなら，適用は行わないなわないようにすれば，容易に制限を実現できる．現ランクの処理中に派生した新たな節同士を合体させる場合は，派生元の情報は両方から伝播させる．そして，ϕ- 関数巻き上げ適用ごとに，すべての派生元情報を調べ，もし，すべての派生元が ϕ- 関数巻き上げ適用不可能の場合だけ，新たな適用を止めるようにし，そうでない場合は，ϕ- 関数巻き上げの適用を行い，適用不可能の派生元情報を削除して，新しい節に伝播させる．こうすることによって，この制約上最大の露出が可能となる．

また，この制限によって露出される計算式の等価性は，6節で議論する最適化の基準，質問伝播によって，冗長を除去したときに，露出されるべき等価性の範囲を含んでいる．

5 値流れグラフの構築

等価性発見アルゴリズムを適用した，拡張値グラフの一つの節には，合体した実際の計算式を登録しておく．この情報によって，等価な計算式 (制御の流れに依存するものも含む) を直接辺で結びあわせていくことができる．このような，等価計算を直接結びあわせたグラフを値流れグラフ [SKR1] [SKR2] という．後に，この値流れグラフ上で，データフロー方程式の解を求めることによって，計算の挿入点の決定と，冗長除去を行う．ここで，データフロー解析に使用する値流れグラフは，各節が対応するフローグラフ節をもっている．ところが，フローグラフは，コード巻き上げのデータフロー解析をブロックする構造をもつものが存在する．2つ以上の後続節をもった節から，2つ以上の先行節をもった節へでている辺が存在する場合である [D] [DP][SKR3]．その辺は危険辺という．以下，危険辺を分割するように基本ブロックを挿入することによって，フローグラフから危険辺は取り除かれているものとする．

値流れグラフの構築は次のように行う．まず，計算式の存在する基本ブックに，等価計算ごとに，節をつくる．その後，ϕ- 関数用の節と辺を挿入する．

この ϕ- 関数の処理は次のように行う．拡張値グラフの節のうち，ϕ- 節のエントリをもつものをとりだし，その ϕ- 節にあたる基本ブロックと，その依存先の方向にあたる先行節に，各拡張値グラフの節に相当する値流れグラフ節を作る．その後，ϕ- 節にあたる値流れグラフ節と，依存先に相当する基本ブロックの先行節の値流れグラフ節とを辺で結んで，それを，値流れグラフの辺とする．

この作業は，異なった等価計算どうしの連結を最初に作っておくことにあたる．この後は，単に，等価計算式を辺で結んでいくだけで，プログラム全体に対する値流れグラフとなる．

等価計算を辺で結ぶ作業は，拡張値グラフの節の中で，プログラム中に元から計算式が存在するものについて，各1回，フローグラフを出口から，うしろ向きに，辿ることによって行う．この際，たどった基本ブロックは，今，対象としている拡張値グラフの節に登録しておく．拡張値グラフにおける，各節への基本ブロックの登録は，7節で述べる計算の巻き上げにおける，飛び越しの問題を解決するために使用する．なお，この基本ブロックをたどって，値流れグラフの構築過程は，次節の制約によって制限される．

6 静質問伝播

値流れグラフの構築の際，同時に，静質問伝播を行う．静質問伝播は，質問伝播という，[RWZ] において，彼らの最適性とともに提案された考えから派生したものである．

2つの等価でない計算が実行され，かつ，(後の制御の流れに依存して) 一方か，他方が，後の計算を冗長にするようなプログラムが存在する．付加的な代入を挟みこまずに，値が使われる場所で，正しい値が格納されるということを保証するかたちで，この冗長を取り除くことはできない．つまり，等価でない計算の定義から，それらの使用へのパスに重なる部分が存在すると，後のレジスタ割り当ての際 (本研究は，後のレジスタ割り付け，割り当てを前提にしている [CJH][RWZ])，同じレジスタにのせることができない．よって，それらの結合部分では，新たに同じレジスタに代入し直す必要があるということである (図7．このような付加的代入を過多に導入しなければならないプログラムが存在する．この場合，付加的代入にかかるコストは，除去される計算のコストを超える可能性がある．

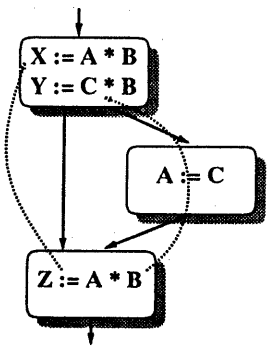

図 7: 質問伝播

[RWZ] では，計算式を冗長として除去するかどうか決めるために，質問伝播を行う．質問伝播は，計算式を置き得るすべての計算点に対して，後ろ向きに，その質問 (計算式) を伝播することによって行われる．特に，次の規則によって，コード巻き上げを制限し，付加的代入の導入を抑える．

- 2つ以上の後続節から異なった，質問が伝播されてきたときは，伝播の源の計算は冗長計算として，除去しない．

本研究では，質問伝播を変形した，静質問伝播を使用する．静質問伝播は冗長かどうかを決めるために行うのではなく，コード巻き上げの範囲を制限するために，値流れグラフの構築範囲を規定する．また伝播の源となる計算は，質問伝播が計算式の挿入可能点，すべてに対して行われるのに対して，静質問伝播は，プログラムに元から存在する計算式が対象である．

静質問伝播は次の規則に従い，値流れグラフ構築とともに，後ろ向きに行う．

- 既に訪問済の基本ブロックで，かつ，等価な計算の値流れグラフ 節があれば，伝播してきた辺を，その節につなぐ．
- 同じ基本ブロックに，2 以上の後続節から伝播してきたとき，その質問が，等価でないなら，辺をプログラムの入り口につなぐ．
- 訪れた節に，等価計算が存在した場合 (ϕ- 関数によるエントリも含めて)，それ以上先への伝播は行わない．

質問伝播が，すべての挿入候補の計算式に対して，試され，最終的な計算式の存在場所に反映されるのにたいし，静質問伝播は対象が元からある計算式であり，より静的な解析といえる．本研究では，この後，値流れグラフにもとづく，データフロー解析によって，計算式の挿入点と，冗長な計算を決定するが，質問伝播とは，除去する冗長の範囲が異なる．

静質問伝播によるコード巻き上げ範囲は，ある計算式 E の異なる質問が同じ基本ブロックに到達する前に，他の計算式 E' が存在するなら，その先行計算式の巻き上げ範囲に依存する．よって，質問伝播によるコード巻き上げ範囲より，広い場合と狭い場合が存在する．

図 8の左側の例は，先行する計算式のために巻き上げ範囲が狭くなる場合を示している．下側の計算式 $C*B$ は，下側の計算式 $A*B$ の巻き上げ範囲に依存してしまい，質問伝播よりも，巻き上げ範囲が狭くなる．ここで，もし，逆に，$C*B$ の伝播を優先して，値流れグラフを構築したとする．ここで，必ずしも，後続の計算が，安全に巻き上げられてくるとは限らない．もし，巻き上がらなかった場合は，$A*B$ は，値流れグラフに沿って，許される以上に巻き上がってしまう可能性がある．

一方，右側の例は，巻き上げ範囲が広くなる場合を示している．先行する下側の $C*B$ によって，

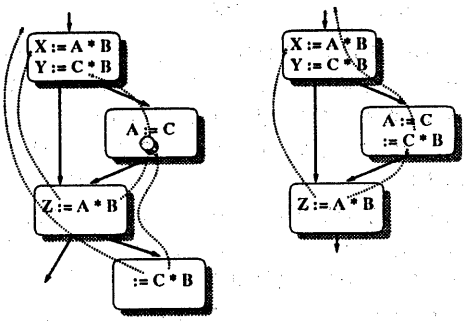

図 8: 静質問伝播の範囲

下側の $A*B$ からつながる値流れグラフは，$A*B$ だけで許される範囲を超えて広がる．

どちらの場合においても，静質問伝播による付加的代入導入によるコストは，冗長計算式除去によって減るコストを超えないことが証明できる．ここで，付加的代入は，後のレジスタ割り付けによってレジスタにのることを仮定しており，レジスタ同士の転送のコストは，如何なる計算式のコストよりも低いと仮定している．

証明． 静質問伝播の範囲は，直接，異なる質問が同じ基本ブロックに到達しない範囲であるから，一つの一時変数にその値を格納することができる．このとき，先行する計算式が冗長計算として，除去され，付加的代入式が導入されたとしても，せいぜい一つに過ぎず，計算除去によるコスト軽減が帳消しになるにすぎない．それより先に導入される付加的代入の導入は，先行する計算式に対して導入されたものであるから，現在対象としている計算式除去に対して，影響しない．∎

7 データフロー解析

7.1 計算点の解析

本研究の最終ステップとして，値流れグラフ上において，データフロー解析を行うことによって，計算式の挿入，冗長計算の除去を行う．データフロー解析による，冗長除去のアルゴリズムは，ほとんどが，Morel と Renvise[MR] の手法がもとになっている [D][DP][SKR3]．そして，そのほとんどが，無駄なコード巻き上げを行わないことによって，予備変数の生存期間を縮め，レジスタにのりやすくするという考えにもとづいている．

本研究で使用するデータフロー方程式は，簡単

のため、単なるコード巻き上げを行うものを採用する。それは[SKR3]の最初にでてくるものと同じである。データフロー方程式は次のようである。

$$D\text{-}SAFE(n) = \begin{cases} false & if\ n = e \\ Used(n) \vee & \\ Transp(n) \wedge \prod_{m \in succ(n)} D\text{-}SAFE(m) & otherwise \end{cases}$$

$$EARLIEST(n) = \begin{cases} true & if\ n = s \\ \sum_{m \in pred(n)} (\neg Transp(n) \vee & \\ \neg D\text{-}Safe(m) \wedge EARLIEST(m)) & otherwise \end{cases}$$

データフロー解析は2種類のデータフロー方程式で一つの形をしている。ここで、s, e, n は基本ブロックを表していて、特に、s は入口節、e は出口節を表している。$D\text{-}SAFE$ は後向きの解析であり、$EARLIEST$ は前向きの解析である。

$D\text{-}SAFE$ は安全に(各パスに新しい計算を導入しない)コード巻き上げをおこなえる範囲を計算する。$EARLIEST$ は安全な計算式挿入のうち最も初期(入口に最も近い)のものを計算する。このデータフロー方程式の解を得ることによって、計算式は最大距離のコード巻き上げが行われる。

このデータフロー方程式を本研究の値グラフ上で使用する場合の解釈は次のようになる。

$$Used(n) = \begin{cases} true & if\ n \text{が計算式を含んでいる} \\ false & otherwise \end{cases}$$

$$Transp(n) = \begin{cases} true & if\ Used(n) = false \\ false & otherwise \end{cases}$$

値流れグラフにおいて、各節は、せいぜい1度更新されるにすぎない。よって、計算量は値流れグラフのサイズに比例し、CN で抑えられる。

7.2 引数定義の飛び越しの解決

ここで行ったデータフロー解析は、各計算式の挿入場所を独立に計算するため、引数の定義が到達しない領域にコードを巻き上げてしまう可能性がある。この問題の解決のために、拡張値グラフを再び使用する。拡張値グラフの各節には、その節に相当する値流れグラフが存在する基本ブロックが登録されている。

$D\text{-}SAFE$ を計算した後、$D\text{-}SAFE$ による巻き上げによって、変化した計算式の使用可能範囲は次のように表される。

元のプログラム中に存在した計算の使用可能範囲 $\bigcup D\text{-}SAFE = true$ の範囲

計算式の使用可能な基本ブロックを n として、$Usable(n) = true$ とすると、各拡張値グラフ節 V の登録された基本ブロックをビットベクタとして、次の関係が成り立つ。

$$D\text{-}Safe_V = D\text{-}Safe_V \wedge \prod_{V' \in children\ of\ V} Usable_{V'}$$

このとき拡張値グラフの節は、ϕ-節しかもたないものを除いて、実行可能関数として扱う。

ランクの低いものから順に、新たな $D\text{-}Safe$ を計算することができ、その計算量は拡張値グラフの辺の数に等しい。

7.3 冗長計算の除去

データフロー解析の結果を利用して、冗長計算を除去し、適切なプログラム点に計算を挿入する。

まず、拡張値グラフの各節ごとに、予備変数 h_i ($for\ i = 1 \cdots n$ 拡張値グラフの節数) を用意する。この後、次の規則に従う。

- $D\text{-}Safe \wedge Earliest$ のとき $h_i := t$ (h_i は対応する拡張値グラフ節から決まる) の挿入
- , 元から存在する計算式は、計算 t を h_i によって置き換える。
- 挿入に使われた計算に依存する ϕ-節を見つけ、ϕ-関数 を挿入する (これを繰り返す)。

挿入した計算式は、各基本ブロックごとに、ランクによってソートする。そして、ϕ-関数のうち、引数に対応する先行節に、その引数を右辺とし、ϕ-関数の代入先を左辺とする代入式を挿入して、SSA形式を通常のプログラム形式に戻す。この段階のプログラムには、多くの異なった変数が存在し、変数を変数に代入するだけのコードも多く導入されているが、この後の、レジスタ彩色手法 [CJH] によって、改善される。

8 まとめ

計算量の面で考えると、過去の研究である大域値番号付けの手法は、仮想変数 n[AWZ][SKR1] に

よって、$O(n^3)$, 値流れグラフを用いるものにおいては、$O(n^4)$ の計算量を要したのにたいし、本研究のアルゴリズムでは、$O(n^2)$ の計算量で実現できる．除去する計算式の範囲を大域値番号付けの最適性と比べると、コード巻き上げの範囲が状況によって異なるため、比較はむずかしいが、計算式の等価性の露出の度合でいえば、本研究はより多くの等価な式を発見できる．また、大域値番号付けと異なり、プログラムの構造にかかわらず、適用することが可能である．

計算量の低さと、除去できる計算式の範囲から、本研究はコンパイラにおける最適化の強力な手法と考えられる．

本稿の手法においては、異なった ϕ- 関数は別の関数として処理しているが、[AWZ] におけるように、ϕ- 関数自身の等価性も考慮に入れることは、重要である．この場合、値流れグラフを、異なった分岐構造を、飛び越して、構築する必要があり、これからの課題といえる．

また、本稿においては、関数の代数的な性質を考慮に入れていない．しかし、最近、冗長除去において、代数的性質を考慮することによって、大きな効果が得られたことが報告されている [BC]．やはり、最近発表された、部分的死亡コード除去 [SKR4] [BC] との組合せもこれからの課題である．

参考文献

[ASU] A. V. Aho, R. Sethi, and J. D. Ullman. Compilers: Principles, Techniques, and Tools. Addison-Wesley, Reading, Mass., 1986.

[AWZ] B. Alpern, M. N. Wegman, and F. K. Zadeck. Detecting equality of variables in programs. Conf. Rec. 15th ACM POPL, 1988, 1-11.

[BC] P. Briggs and K. D. Cooper. Effective Partial Redundancy Elimination. SIGPLAN '94 ACM Conf. PLDI, 1994, 159-170.

[CJH] Fred C. Chow and John L. Hennessy. The Priority-Based Coloring Approach to Register Allocation. ACM TOPLAS, Vol.12, No.4 (Oct. 1990), 501-536.

[D] D. M. Dhamdhere. Practical Adaptation of the Global Optimization Algorithm of Morel and Renvoise. ACM TOPLAS, Vol. 13, No.2 (Apr. 1991), 291-294.

[DP] D. M. Dhamdhere and H. Patil. An Elimination Algorithm for Bidirectional Data Flow Problems Using Edge Placement. ACM TOPLAS, Vol. 15, No. 2 (Apr. 1993), 321-336.

[MR] E. Morel, and C. Renvoise Global Optimization by Suppression of Partial Redundancies. Commun. of the ACM 22, 2 (1979), 96 - 103.

[RL1] J. H. Reif and H. R. Lewis. Symbolic evaluation and the global value graph. Conf. Rec. 4th ACM POPL, 1977, 104 - 118.

[RL2] J. H. Reif and H. R. Lewis. Efficient Symbolic Analysis of Programs. Journal of Computer and System Sciences. 32,1986, 280-314.

[RWZ] B. K. Risebm, M. N. Wegman and F. K. Zadeck. Global value numbers and redundant computations. Conf. Rec. 15th ACM POPL, 1988, 12 - 27.

[SKR1] B. Steffen, J. Knoop and O. Ruthing. The value flow graph: A program representation for optimal program transformations. In Proceedings 3rd ESOP, Copenhagen, Denmark, Springer-Verlag, LNCS 432 , 1990, 389 - 405.

[SKR2] B. Steffen, J. Knoop and O. Ruthing. Efficient code motion and an adaption to strength reduction. In Proceedings 4^{th} TAPSOFT, Brighton, United Kingdom, Springer-Verlag, LNCS 494 1991,394-415.

[SKR3] J. Knoop, O. Rüthing and B. Steffen. Lazy Code Motion. SIGPLAN'92 ACM Conf. PLDI, 1992, 224-234.

[SKR4] J. Knoop, O. Rüthing and B. Steffen. Partial Dead Code Elimination. SIGPLAN '94 ACM Conf. PLDI, 1994, 147-158.