非循環グラフにおける支配関係の簡潔な検出算法

齋藤鐵男[†] 鈴木貢[††] 渡邊坦[††]

本稿では，非循環有向グラフ（DAG; Directed Acyclic Graph）で表される制御フローグラフに対し，その支配関係を求める簡潔な算法を報告する．この算法は，頂点数 N，辺数 M のグラフに対して，大きさ N のスタックのプッシュとポップに基づき，$N-1$ 回のリダクションを繰り返す．リダクションの進行中に，次のリダクション対象頂点を探す手続きはまったく必要としない．支配木と支配辺境の算出は，グラフに対する一連のリダクション操作がすべて終了したとき同時に完了する．本算法は計算量が $O(N)$ の部分と $O(M)$ の部分からなり，前者についての計算量は算法からほとんど自明である．後者については，N および M に対して独立な，グラフの構造の差異による若干の変動はあるものの，算法および計測の結果から，実用のプログラムでは，ほぼ M に比例し，$O(M)$ であることが主張できる．SpecCFP92 テストプログラムから約 240 本を選抜して実行した結果は，統計的に，M に対して，多少のばらつきをともなう線形性を示し，比例係数は平均 1.5，最大 2.5 であった．これは既発表の他の方法より約 3 倍高速である．

A Simple Algorithm to Identify Dominance Relations in DAG

Tetsuo Saitou,[†] Mitsugu Suzuki[††] and Tan Watanabe[††]

In this paper, we report a compact algorithm for dominance analysis of control flow graph in the form of directed acyclic graph (DAG). This algorithm repeats N-1 reductions on the graph with N nodes and M edges, controled by push and pop of a stack of size N. There is no need of any search for next node to be reduced in the progress of reduction. The dominator tree construction as well as dominance frontier computation is completed at the end of the series of reductions for the graph. The algorithm has two sections of O(N) and O(M) complexity. The complexity of former is self-evident from the algorithm. The complexity of latter can be declared to be almost linear and O(M) for real programs from the algorithm and observation. Although we have a little variance from linearity, it is caused by the difference of graph structure which is independent of N or M. Our test using about 240 samples of real programs from SpecCFP92 resulted in scatter-diagram suggesting its linearity about M, and showed 1.5 M in average and 2.5 M at maximum, statistically for the O(M) section. This means that our algorithm is about 3 times faster than another one that has been published already.

1. はじめに

制御フローグラフにおける支配木や支配辺境の検出は，プログラム構造の解析や最適化の分野で，最も基礎的で広い用途のある問題である[14]．たとえば支配木はループ構造の本体の検出に，支配辺境は単一代入形式におけるデータ定義の適合化点の検出に用いられ，ともに言語処理系における最適化技法の中で利用される．そのため，できるだけ高速な処理の実現に向けてアルゴリズムが研究されてきた．最近は JIT（Just-In-Time）コンパイラが実用化し，高速性に加えて簡潔なアルゴリズムの研究が必要である．本稿では簡明なアルゴリズムを提示し，それに基づいて高速性を保つ実装を試みた．

制御フローグラフの支配木や支配辺境を検出する算法については多くの発表があり，最近の研究では難度の高いリダクション不能なグラフ（irreducible graph）に視点が向けられている．しかし実際のプログラムではリダクション可能なグラフ（reducible graph）が圧倒的に多く，Fortran77 以後の構造化言語ではリダクション不能なグラフの発生する可能性が特に低い．また構造化言語の出現以前でも，ほとんどのプログラムはリダクション可能なグラフ構造になっているという意見もある[7]．最近の研究[15]では，リダクション不能

† 電気通信大学電気通信学研究科情報工学専攻
Department of Computer Science, Graduate School of Electro-Communications
†† 電気通信大学情報工学科
Department of Computer Science, The University of Electro-Communications

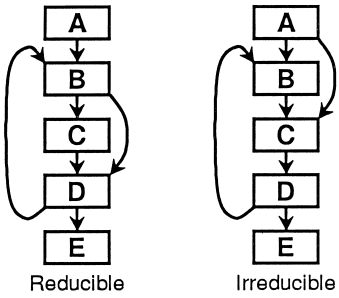

図 1 制御フローグラフ
Fig. 1 Control Flow Graph.

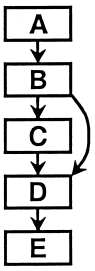

図 2 非循環有向グラフ
Fig. 2 DAG: Directed Acyclic Graph.

なグラフを，等価なリダクション可能なグラフに変換し，さらに非循環グラフ（acyclic graph）に変換して支配関係の解析を行う方法が発表されている．リダクション可能なグラフの戻り辺を隠蔽した非循環グラフは，もとのリダクション可能なグラフと同じ支配関係を保っている[7),14)] ので，支配木や支配辺境の検出に都合のいい構造として重要である．

本稿では，サイクルのない制御フローグラフの支配木と支配辺境の検出（以下，支配関係解析と呼ぶ）を行う簡潔で高速な算法を提示し，実プログラムから取得した約 240 件のグラフをサンプルとして算法の計算量を実測した結果を報告する．この算法ではリダクションを利用して，直接支配頂点を検出し，逐次支配木の情報を収集する．算法が収束したとき，支配木の全情報とともに，頂点ごとの支配辺境の情報が完成する．この中で，リダクション対象の頂点の番号をトポロジカルオーダでスタックに積み，かつ取得する簡単な方法で，リダクション継続中のリダクション対象頂点の探索を無用化し，計算量を極小化するとともに算法の簡潔化を実現した．この算法の計算量はグラフの辺の数 M に対して $O(M)$ である．

制御フローグラフとは，プログラムの制御の流れを表すグラフで，基本ブロックを表す頂点を，制御の流れを示す辺で結んだ有向グラフをいう．制御フローグラフはただ 1 つの入口の頂点を持つ．**図 1** に 2 種類の制御フローグラフを示し，その違いを以下に述べる．

繰返しループは，制御フローグラフの中で後述のサイクルを形成する．図 1 の右の図のように，サイクルへの入口が 2 カ所以上のとき，そのサイクルはリダクション不能（irreducible）といわれ，これを含む制御フローグラフもまた，リダクション不能といわれる．実際には，手法によってはループの認識が一義的に定まらない[12)] 等の不具合がある．構造化言語の出現以前の初期の頃には，IF と GOTO を用いてループを構成し，この種の構造が無制限に現れうる環境があったが，構造化プログラミングの思想が普及した現在は，リダクション不能なサイクルを含む実用プログラムは少ない．この構造に対する支配関係解析の方法として，制御フローグラフをリダクション可能なグラフに変換する提案もある[15)]．

図 1 の左の図のように，サイクルの入口が 1 つの場合はリダクション可能（reducible）といわれ，実在するほとんどのプログラムはこの範疇に属する．以下，本稿ではリダクション可能を**可約**，リダクション不能を**非可約**と略称する．可約なグラフのサイクルは，深さ優先探索で検出される戻り辺（back edge）と関係づけて認識できる．これらの戻り辺を無視し，または削除することにより，サイクルのないグラフを得ることができる．サイクルのない制御フローグラフは 1 つの入口頂点を持つ連結した非循環有向グラフであり，**DAG**（Directed Acyclic Graph）と呼ばれる．図 1 の左のグラフ（Reducible）の戻り辺を除いた DAG は**図 2** のようになる．可約なグラフの戻り辺を除いて得られる DAG における支配関係は，もとのグラフの支配関係に等しい[7),14)] ので，支配関係解析には DAG を利用する．本算法は，可約なグラフを DAG に変換して支配関係解析を行うことを前提に，DAG における支配関係解析を目的とする．DAG への変換自体[7),14)] は主題から外れるので扱わない．

以下では DAG を $G=(V,E,e)$ で表し，単にグラフと呼ぶ．ここに，V はグラフの頂点の集合，E はグラフの辺の集合，$e \in V$ はグラフの入口頂点である．$N=|V|$ は**頂点の数**，$M=|E|$ は**辺の数**を表す．$u,v \in V$ に対して u から v への辺が存在するとき，その**辺**を $u \rightarrow v$ で表し，u を v の**直接先行頂点**，v を u の**直接後続頂点**と呼ぶ．以下では，直接先行頂点，直接後続頂点を，それぞれ**先行頂点**，**後続頂点**と略称する．頂点 x の**後続頂点の集合**を $S(x)$，**先行頂点の集合**を $P(x)$ で表す．$|S(x)|, |P(x)|$ は，それぞれ頂点 x の後続頂点の数，先行頂点の数である．$G=(V,E,e)$ を $G=(V,S,e)$ とも書く．ここで S は $\{S(x)|x \in V\}$

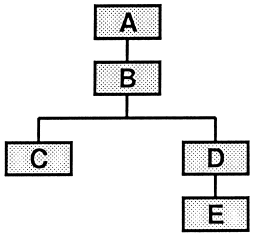

図 3 支配木
Fig. 3 Dominator tree.

である．辺の列 $x_0 \to x_1, x_1 \to x_2, ..., x_{k-1} \to x_k$ を**経路**と云い，$x_0 \longrightarrow x_k$ で表す．経路 $u \longrightarrow v$ が存在するとき，v は u から**到達可能**であるという．経路 $x_0 \longrightarrow x_k$ が存在して $x_0 = x_k$ のとき，この経路を**サイクル**と呼ぶ．DAG にはサイクルがないので，経路上に同じ頂点が複数回現れることはない．連結したグラフでは，すべての $w \in V$ は e から到達可能である．e から w へのすべての経路に頂点 v が含まれ，$v \neq w$ のとき，頂点 w は頂点 v に**支配される**という．このとき v を w の**支配頂点**と呼び．v **dom** w と表す．v **dom** w であり経路 $v \longrightarrow w$ 上に v 以外に w の支配頂点が存在しないとき，v を w の**直接支配頂点**と呼び，v **idom** w または **idom**$(w) = v$ と表す．

辺 **idom**$(w) \to w$, $\{w | w \neq e, w \in V\}$ が構成する木は G の**支配木**と呼ばれる．**図 3** は図 2 のグラフの支配木を示す．これは図 1 の左のグラフの支配木でもある．この木では頂点 C，D はともに頂点 B の子であり，グラフ上では共通の直接支配頂点 B を持っている．支配木の根は e である．支配木における e 以外の頂点 v の親は **idom**(v) である．**idom**$(v) = u$ のとき，支配木における対 (u, v) を支配木の**枝**と呼ぶ．頂点 $x \in V$ の**支配辺境**とは，x の後続頂点または x の支配を受ける頂点の後続頂点のうち，x の支配を受けない頂点の集合である．図 1 の左のグラフ，図 2 のグラフのいずれにおいても，頂点 D は頂点 C を通らない入口からの経路があるので，頂点 C の支配辺境の頂点である．頂点の集合 $X = \{x | x \in V\}$ に対して，x の支配辺境の直和を集合 X の支配辺境という．

2. 関 連 研 究

最初にこの問題が提起されたのは，Lowry ら (1969) [1] によるとされている[13]．Aho らはグラフの頂点を逐一削除して，それぞれに対してどの頂点がグラフの入口から到達可能かを調べるという方法で，$O(N(N + M))$ 時間の算法を提案した (1972) [2]．引き続き，Aho らは，可約なグラフに対して，木構造の中で**最近共通祖先**（nearest common ancestors）を求める方法で，$O(N + MlogM)$ 時間の算法を示した (1973) [3]．最近共通祖先とは，木の任意の複数の頂点に対して共通な祖先のうち最もルートから遠い頂点をいう．一方，Tarjan は直和計算や待ち行列の効率的な取扱いを考案し，深さ優先探索の情報から支配木を計算する $O(NlogN + M)$ 時間の算法を発表した (1974) [4]．可約なグラフについては，Ochranova がリダクション操作で支配木を求める算法を提案し，計算量は $O(M)$ と述べている (1983) [5]．しかしこの論文には計算量の議論がなく，“The time complexity of our algorithm is surely not worse $O(N^2)$ and a detailed analysis indicates that it equals $O(M)$. At least, no counterexample was found.” との記述がある[5]．計算量の議論がないことから，“$O(M)$” の主張に対して疑問を呈している論文[13] もある．近年では，Ramalingam らが，制御フローグラフの辺の増減に対してインクリメンタルに支配木を修正する問題を論じ，その中で DAG の支配木を求める方法として，最近共通祖先による方式を用いると述べている (1994) [10]．Ramalingam はその後，非可約なループを含む一般のグラフ構造について，抽象化したループ検出のフレームを提案し，最終的に DAG に変換して支配木を作る方式を示した (1999) [15]．その中で，DAG の扱いに関しては，Gabow [8] のデータ構造を用いた最近共通祖先の計算により，$O(N)$ で計算できると述べている．また，同年，Alstrup らは一般の制御フローグラフについて，Lengauer らの算法を改善した方法[13] を提示した．その中で可約なグラフについては，論文 5) の方式の計算量が $O(M)$ であることに疑問を呈し，別途，最近共通祖先の計算を用いて可約なグラフの支配木を求める $O(N + M)$ 時間の算法を示した．

最近共通祖先を用いる直接支配頂点の計算のアルゴリズムは質，量ともに最近共通祖先の計算に依存する．その内容[6] は対称な完全 2 分木における任意の 2 頂点の最近共通祖先が $O(1)$ で求めうるという事実に基づくものであるが，任意の木 T における最近共通祖先を求めるためには，T を完全 2 分木に関係づける必要から，中間構造として圧縮木（compressed tree）と平衡木（balanced tree）を必要とする．これらのデータ構造は，本稿の方式で用いる汎用的な構造を超える複雑性を持ち，内容的に本稿の方式と同程度に簡潔な実装は困難である．

支配関係解析に関して多くの研究が発表されているが，大きな流れは，計算量の改善を主題とするものであり，可約なグラフに限らず，一般制御フローグラフを視野に入れている．一方可約なグラフ，特に DAG

についての研究や記述については，Ochranova がリダクションによる方法[5] を発表したほかは，最近共通祖先の計算についての発表[6),8)] や，この方法によると記述するにとどめた論文が見受けられる．本稿は，実用プログラムのほとんどが可約なグラフであることから，対象を可約なグラフに限定し，原理的に明解なリダクションによる算法の簡潔化と高速化を追求した．

3. 算　　法

DAG の支配木と支配辺境を求める本算法を，**図 6** に示す．図の左端に並ぶ 2 桁の数字は説明用の行番号である．

頂点に関する情報は頂点ごとの構造体の配列で表現される．簡単のためこれらの頂点は深さ優先探索の後付け番号順に並んでいるものとする．この番号は DAG の頂点についてのトポロジカルオーダ順の番号であり，深さ優先探索の逆順後付け番号を用いる．以後，この番号を**DFSN**，この順番を**DFSN 順**（Depth First Search Numbering order）と呼ぶ．以後，頂点名は DFSN を用いているものとする．この配列の要素である頂点は，リダクションにつれて削除したりせず，頂点の連結情報，すなわち先行頂点数や，後続頂点集合の番号のリストの内容を，グラフの変化に対応して更新する．支配木は，頂点がリダクションされる時点での先行頂点の番号を記録することで表す．

支配木は**図 5** の下の図のように必ずしも連結でない部分木として成長し，リダクションの完了とともに全体が完成する．

リダクションされた頂点は，先行頂点の後続頂点集合のリストからその番号を消されることで，グラフから解放される．図 5 の上の図の (2) では頂点 B の後続頂点集合が { C, D } であり，頂点 C がリダクションされた (3) では，B の後続頂点集合から C が除かれて { D } に変わっている．ここで，スタックにあった C の代わりに D が現れているのは，C のリダクションによる後続頂点集合の直和の計算において，D の先行頂点数が 1 になり新規の候補頂点としてプッシュされたことを示す．

また，リダクションされた頂点の情報は，リダクション直前の状態で保存される．図 5 では，頂点 C は (3) でグラフとの接続がなくなるが頂点 C の後続頂点集合 { D } は (2) で頂点 C がリダクションを受ける直前の状態 { D } のまま保存されることが (3) での頂点 C の表示で示されている．この時点以後頂点 C の後続頂点 { D } は変化せず，頂点 C の支配辺境を示すものとなる．

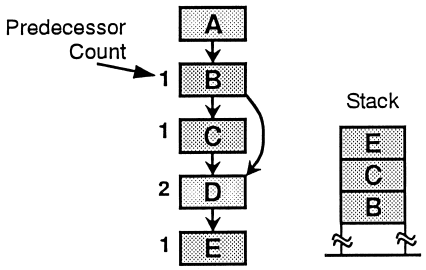

図 4 DFSN，先行頂点数，スタック
Fig. 4 DFSN, predecessor count and stack.

図 5 で，有向グラフの頂点に対応した中括弧 { } の内容は，各頂点の，そのときのグラフでの後続頂点の集合を表し，有向グラフの下に分離して描かれた頂点のそれは，各頂点の支配辺境を表している．図ではこれを区別するために点線で囲んで示した．

リダクションの進行を制御するスタックには，頂点の実体ではなく，頂点の番号がプッシュとポップを受ける．いわばダイナミックなインデクステーブルである．図 5 で，各グラフの左に模式図で示したスタックの内容は下から上に向かって DFSN 順に並んだ候補頂点の番号を意味する．(1)〜(4) の各段階でスタックの上端から取り出される番号の頂点がリダクションされ，逐次右の図に状態が移りかわる．

3.1 初期設定と主要データ

行 13〜18 はフラグテーブル，先行頂点数，スタックの初期設定である．

フラグテーブルは行 09 の配列 *flags* で，頂点集合の直和の計算において共通要素を検出するために用いる．初期設定ではテーブル全体に，DFSN ではありえない値，たとえば −1 を埋め込む．

先行頂点数は文字どおり頂点ごとの先行頂点の数であり，初期設定では DFSN 順にアクセスして，後続頂点の先行頂点数をインクリメントする．**図 4** のグラフの左の数字はそれぞれの先行頂点数を示す．

スタックは行 08 の配列 *stack* である．初期設定ではリダクション候補頂点の番号を，DFSN 順にスタックに格納する．図 4 のスタックは，図 2 のグラフに対するスタックの初期状態を示している．リダクション候補頂点とは，現在取扱い中のグラフにおいて，先行頂点が単一である頂点である．一般にはこの条件が成り立てばその頂点はリダクションが可能であるが，本算法では支配木と支配辺境の検出を目的とするため，リダクション候補頂点のうち，DFSN 順で最後の頂点（スタックからポップした番号の頂点）を，毎回のリダクション対象頂点とする．以下では，リダクション候補頂点，リダクション対象頂点を，それぞれ**候補頂**

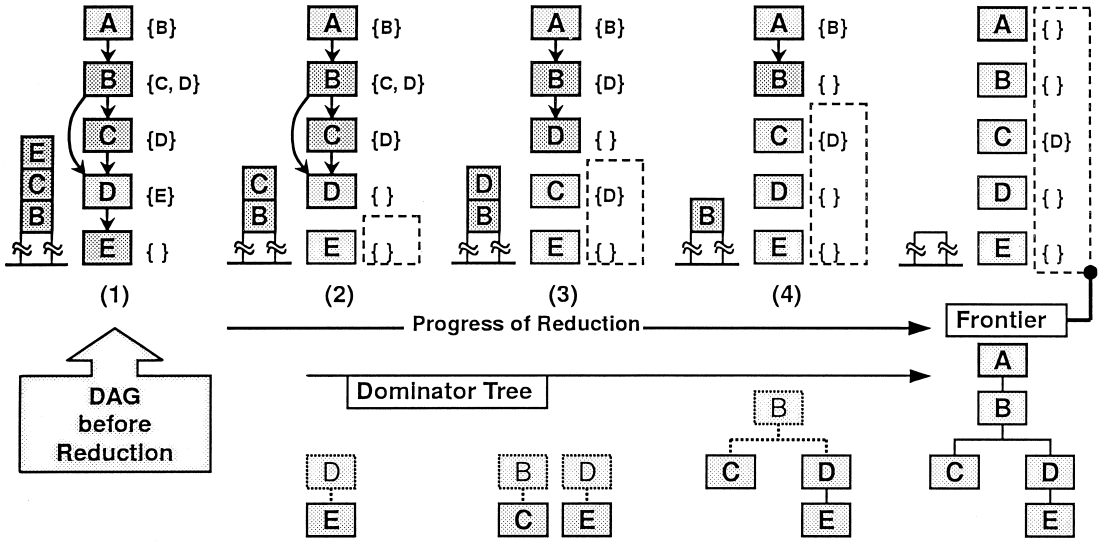

図 5 リダクションの過程
Fig. 5 Process of reduction.

点，**対象頂点**と略称する．また対象頂点の単一の先行頂点を，**対象–先行頂点**と呼ぶ．

3.2 リダクションとスタック

ここではスタックを利用したリダクション操作の概要を説明する．まず，対象頂点の取り出しについて述べる．スタックからポップした番号の頂点は DFSN 順で候補頂点の最後の頂点である．リダクションのループの始めにスタックからポップした番号の候補頂点 y は，他の頂点を支配していない（支配木では葉の位置にある）．もし他の頂点を支配しているならば，y よりも DFSN の大きい頂点 z で候補頂点の条件を満たすものがあり，スタックの最後は y ではありえない．リダクションによって変化するグラフの中にある候補頂点が，つねにスタック上にあり，スタックの内容が DFSN 順になっていれば，このことはつねに成り立つ．したがって，ポップした頂点を無条件に対象頂点としてリダクション操作を実行しても，グラフの支配関係を乱すことはない．

次に，候補頂点番号のスタックへの追加について述べる．グラフの初期状態で，すべての候補頂点の番号がスタックに積まれるのは，グラフにまったく分岐のない特別な場合のみで，普通は候補頂点の一部の番号のみがスタックにある．その他の頂点はリダクションの進行につれて，次の状況で検出される．リダクションにおいて後続頂点の直和を計算中，共有頂点があれば，その先行頂点数が 1 つ減る．その結果先行頂点数が 1 になれば，新しい候補頂点が検出されたことになる．この頂点の番号は検出時点でスタックにプッシュすればよい．共有頂点がない場合はスタックの要素数が対象頂点 1 つ分減少する．

スタックを使う目的は，つねに DFSN の逆順に候補頂点のリダクションを実行できるように保証することである．これにより，グラフの支配関係を保存しながらリダクションを続けられる．加えて，対象頂点の番号の取り出しや新規候補頂点の番号の格納を，1 つのスタックに集中して，そのポップとプッシュで実行するので，候補頂点選択のために頂点集合を探索する手続きがいっさい不要になり，計算量を軽減し算法を簡素化できる．

DAG はつねに可約であり，対象頂点は必ず 1 回リダクション可能になるはずなので，入口頂点を除くすべての頂点の番号は必ず 1 回，かつ 1 回限りスタックにプッシュされ，ポップを受ける．リダクションは，行 19 から行 35 までの範囲の手続きを，スタックが空になるまで繰り返すことで実行される．スタックが空になったとき，グラフは入口頂点 1 個に縮退し，入口頂点を根とする支配木が完成する．

また，リダクションが終了したとき，各頂点の後続頂点集合は，各頂点のリダクション時点の内容，すなわちそれぞれの頂点の支配辺境となって残っている．

3.3 算法の詳細

以下，図 6 の各行を説明する．

リダクションのループ

行 19 の while の下のループ本体（行 20〜40）が 1 頂点のリダクションおよびそれに関係する操作のいっさいを実行する．

```
01  procedure reduce_DAG
02  Input:  Ga= (V, S, e) ; DAG : V is the set of vertices, e the entry
03                        ; S(x) is the set of immediate successors of x.
04                        ; Each vertex is named by the post DFS number.
05  Output: T             ; the dominator tree of Ga
06  Local:  x,y           ; vertices: x → y is the edge to be reduced
07          v,u           ; successor u ∈ S(x), v ∈ S(y)
08          stack[]       ; stack of vertices (for x → y) to be reduced
09          flags[]       ; flags[z] is ON if vertex z exists in S(x).
10          v.Npred       ; the count of immediate predecessors of v
11
12  begin
13      Clear flags, v.Npred(for all v ∈ V), and stack ;
14      for( all y ∈ V : by DFSN order ) do
15          begin
16              for( all v ∈ S(y) )  do Increment v.Npred ;
17              if( y.Npred == 1 )  then push x → y into stack ;
18          end
19      while( stack is not empty ) do
20          begin
21              pop x → y from stack ;
22              add y to T as a child of x ;
23              for( all u ∈ S(x) ) do
24                  begin
25                      if( u == y ) then
26                          Delete u from S(x) ; [located in S(x) already]
27                      else
28                          flags[u] := y ;
29                  end
30              for( all v ∈ S(y) ) do
31                  begin
32                      if( flags[v] == y )  then
33                          begin
34                              v.Npred := v.Npred-1 ;
35                              if( v.Npred == 1 )  push x → v to stack ;
36                          end
37                      else
38                          Add copy of v to S(x) ;
39                  end
40          end
41  end
```

図 6 支配木と支配辺境の検出
Fig. 6 Identifying dominator tree and dominance frontier.

対象頂点の取得と支配木への記録

まず行 21 でスタックから対象頂点 y をポップして，行 22 で y を支配木に x の子として無条件に追加する．y の先行頂点 x は，初期設定では先行頂点数のカウントとともに頂点ごとに保存する．全頂点についてカウントが終了したとき，カウントが 1 の頂点についてこの先行頂点番号が有効になる．リダクション中に発生する候補頂点についてはそのときの対象先行頂点が新しい候補頂点の先行頂点である．

リダクション

次に対象頂点 y のリダクション操作を実行する．リダクション操作とは対象頂点 y をグラフから隠すことである．そのために，対象–先行頂点 x の後続頂点集合から対象頂点 y を除き，対象頂点 y の後続頂点集合を対象–先行頂点 x の後続頂点集合に併合する（直和を作る）．詳細は以下のように，大別して行 23〜29 に示す対象–先行頂点 x の $S(x)$ についての操作と，行 30〜39 に示す対象頂点 y の $S(y)$ に関わる部分に分けられる．

対象–先行頂点についての操作

行 23〜29 は，直和を受け入れる対象–先行頂点 x の後続頂点 $S(x)$ についての計算である．すべての $u \in S(x)$ を調べ，$u = y$ ならば，行 26 で対象頂点 u を集合 $S(x)$ から削除し，$u \neq y$ なら，行 28 で $u \in S(x)$ について，フラグテーブル上の DFSN 相当位置 $flags[u]$ に対象頂点の値 y を書き込む．このテーブルは $S(y)$ を $S(x)$ に併合する際に，共有頂点を検出するためのフラグテーブルである．$u \in S(x)$ の DFSN 相当位置 $flags[u]$ に値 y を格納しておき，$v \in S(y)$ の DFSN 相当位置 $flags[v]$ の値が y ならば，v は $S(x)$ にも含まれている．対象頂点固有の値 (DFSN) y は全リダクション過程を通じて今回のリダクション以外に現れることがないので，このフラグテーブルは初期化操作なしに継続使用できる．行 23 のループで集合を走査し，行 25 の条件で削除操作を行うので，削除対象要素 u の $S(x)$ 内での位置はすでに認識されていて，あらためて探索する必要はない．

対象頂点についての操作

行 30〜39 は，リダクション対象頂点 y の後続頂点 $S(y)$ についての操作である．$v \in S(y)$ として，行 32 でフラグテーブルの該当位置 $flags[v]$ が対象頂点の値 y に等しいか否かを調べることにより，$S(x)$ の探索なしに $S(x)$ と $S(y)$ との v 共有の有無を判定する．

共有頂点の処理

行 27〜31 は，共有している場合の操作である．この場合は $S(x)$ に，すでに $v \in S(y)$ と同じ頂点があり，v をあらためて $S(x)$ に加える必要はない．しかし，頂点 v の先行頂点数 $v.Npred$ については，リダクション前にはこの頂点 v の先行頂点として対象–先行頂点 x と対象頂点 y があり，リダクション後はその 1 つの頂点 y が消えるので，行 34 により先行頂点数 $v.Npred$ を 1 つ減らす必要がある．この状態が起きるのは，先行頂点数が 2 以上の場合に限られるから，$v.Npred$ を減らした結果は 0 またはそれ以下になることはない．行 35 の判定で v の先行頂点数が 1 になっていれば，頂点 v を新しい候補頂点としてスタックにプッシュする．このとき候補頂点 v の対象–先行頂点は明らかに x である．また，この結果のスタックの並びは依然 DFSN 順である．1 つの対象頂点 y のリダクション中に，後続頂点集合 $S(y)$ に属する複数の頂点が新規に候補頂点と判定されてスタックにプッシュされる場合，それらはいずれも DFSN 順で対象頂点 y の後続要素であるし，ここでプッシュされる頂点相互間では DFSN 順が同等であるので，ここで $v \in S(y)$ から検出した候補の番号を検出順に無条件にスタックにプッシュしても，スタックの要素の DFSN 順を損なうことはない．

非共有頂点の処理

行 37〜38 は，共有していない場合の操作であり，単純に対象頂点 y の後続頂点 v を対象–先行頂点 x の後続頂点の集合 $S(x)$ に追加する．この場合，新に $S(x)$ に加わった $v \in S(y)$ の先行頂点からは y が消えて x が加わるので，先行頂点数 $v.Npred$ は結果的にそのままでよい．

支配辺境の形成

以上の操作を，スタックからポップした番号の頂点を y として実行することを，スタックが空になるまで繰り返すと，各頂点の後続頂点集合の情報は，その頂点をリダクションする直前の状態で，それぞれの頂点の支配辺境を表す情報として残る．

頂点 y をポップして頂点 x にリダクションするとき，x **idom** y であり，$z \in S(y)$ は y **idom** z ではない．z はトポロジカルオーダで y よりも後にあるから，もし y **idom** z ならば，y の前に z がリダクションされるはずである．y **idom** z でないので，$z \in S(y)$ は y の支配辺境の頂点である．

リダクション過程の任意のグラフにおける頂点の直接支配頂点は，初期のグラフでも同じ頂点の直接支配頂点である☆．いいかえると，リダクション過程の任意の時点の支配木は初期のグラフの支配木の部分木で

☆ **定理 6** 参照.

ある．したがって，リダクション過程の任意のグラフにおける頂点の支配辺境は，初期のグラフでも同じ頂点の支配辺境である．

4. 定　　理

本算法を支える定義，公理，定理を以下に述べる．グラフ $G=(V,E,e)$ は DAG である．

定義 0　リダクション $R(G,x\rightarrow y)$ とは，グラフ $G=(V,E,e)$ の頂点 $x,y\in V$ において，$P(y)=\{x\}$ であって，$S(y)=\phi$ であるとき，または $S(y)\neq\phi$ ならばすべての $z\in S(y)$ について $|P(z)|>1$ であるとき，x，y および辺 $x\rightarrow y$ を頂点 x に置き換え，$S(x)=S(y)\cup(S(x)-y)$，とすることをいう．

公理 1　グラフ $G=(V,E,e)$ の頂点 $x,y\in V$ において，$P(y)=\{x\}$ ならば $\mathbf{idom}(y)=x$ である．

定理 2　グラフ $G=(V,E,e)$ で，$x\in V$ が $S(x)=\phi$ であるか，または $S(x)=\{y_1,y_2,...,y_n\}\neq\phi$ で，すべての$k\,(1\leq k\leq n)$について $|P(y_k)|>1$ ならば，$\{z\,|\,x\,\mathbf{dom}\,z,\ z\in V\}=\phi$ であり，したがってまた，$\{z\,|\,x\,\mathbf{idom}\,z,\,z\in V\}=\phi$ である．

証明　$S(x)=\phi$ ならば x が支配する頂点がないことは明らかである．$S(x)\neq\phi$ ならば，$x\ \mathbf{dom}\ z$ なる $z\in V$ があると仮定する．このとき e から z に至る経路はすべて x を通り，したがって x の直接後続頂点 $y_k\in S(x)=\{y_k\,|\,x\rightarrow y_k,1\leq k\leq n\}$ を通る．$|P(y_k)|>1$ であるから，y_k に対して辺 $x\rightarrow y_k$ を含まない経路 $e\longrightarrow y_k$ が 1 つ以上ある．そのいずれかが x を通らなければ，$x\ \mathbf{dom}\ z$ でない．これは仮定に反するから，経路 $e\longrightarrow y_k$ は x を通り，したがって x の直接後続頂点 $y_j\in S(x)=\{y_j\,|\,x\rightarrow y_j,1\leq j\leq n\}$ のいずれかを通る．すなわちすべての $y_k\in S(x)$ について，辺 $x\rightarrow y_k$ とは別に入口から y_k に至る経路があり，その経路すべてが $S(x)$ に属するいずれかの頂点を通る．すなわちこれらの経路の中で $S(x)$ に属するいずれの頂点をも通らない経路はない．したがってこれらの経路は，$S(x)$ に属するすべての頂点，もしくは一部の頂点を含むサイクルを形成し，グラフが DAG であることに矛盾する．よって $x\ \mathbf{dom}\ z$ なる $z\in V$ はありえず，$x\ \mathbf{idom}\ z$ なる $z\in V$ もまたありえない．□

定理 3　$x\in V$，$W=\{z\,|\,x\,\mathbf{idom}\,z,z\in V\}\neq\phi$ ならば，$W\supseteq\{y\,|\,P(y)=\{x\},\,y\in V\}\neq\phi$ である．

証明　x は直接支配頂点なので $S(x)\neq\phi$ である．もし $P(y)=\{x\}$ なる $y\in V$ が存在しないとすれば，すべての $y\in S(x)\neq\phi$ について $|P(y)|>1$ である．したがって定理 2 から，$x\ \mathbf{idom}\ z$ となる頂点 z は存在しないことになり，$W=\{z\,|\,x\,\mathbf{idom}\,z,\,z\in V\}\neq\phi$ の仮定に反する．□

定理 4　$G'=R(G,x\rightarrow y)$ ならば，G' の支配木は G の支配木から枝 (x,y) および頂点 y を除去した木に等しい．

証明　リダクション $R(G,x\rightarrow y)$ の定義における $x,y\in V$ に関する仮定と定理 2 から，y は直接支配頂点とはならない．したがって，G の支配木において y は葉であるから，リダクション $R(G,x\rightarrow y)$ によるグラフ G の頂点 y と辺 $x\rightarrow y$ の除去は，G の支配木において頂点 y と枝 (x,y) の除去以外の変化を生じない．□

定理 5　グラフ $G=(V,E,e)$ に対して，$G_0=G,G_1=R(G_0),...,G_k=R(G_{k-1}),k=|V|-1$ かつ $G_k=(V_k,E_k,e)$ が $V_k=\{e\},E_k=\phi$ であるようなリダクションの系列 $G_0,G_1,...,G_k$ が存在する．

証明　グラフ G の支配木から始めて，葉 y と，枝 (x,y) に対応する G の頂点 y と辺 $x\rightarrow y$ に逐次 $R(G,x\rightarrow y)$ を適用すれば，**定理 4** により適用のつど支配木から葉 y と，枝 (x,y) が除去され，他の支配関係は不変のまま，グラフ G において頂点 y と辺 $x\rightarrow y$ が除去される．$R(G,x\rightarrow y)$ は支配木の葉がある限り適用可能で，支配木は $G=(V,E,e)$ において $V=\{e\},E=\phi$ となるまでは，つねに葉を持っている．したがってこの過程は，グラフが支配木のルートである e に縮退するまで継続して，系列 $G_0,G_1,G_2,...,G_k$ を与える．条件 $E=\phi$ により繰返しは収束し，終了時点のグラフを $G_k=(V_k,E_k,e)$ とすれば $V_k=\{e\},E_k=\phi$ である．□

定理 6　グラフ $G=(V,E,e)$ に対する，**定理 5** のリダクション系列 $G_0,G_1,...,G_k$ 中，いずれかのグラフ $G_i(0\leq i\leq k)$ において $x,y\in V$，$P(y)=\{x\}$ ならば，グラフ G で $x\ \mathbf{idom}\ y$ である．

証明　定理 4 により，リダクション系列におけるリダクションの操作 $R(G,x\rightarrow y)$ は，支配木において葉 y と枝 (x,y) を除去し，グラフでは頂点 $x,y\in V$ と辺 $x\rightarrow y$ を頂点 x に縮退するだけで，グラフの他の部分には何等変化をもたらさない．したがって，最初のグラフで $x\ \mathbf{idom}\ y$ であれば，系列のいずれかのグラフで，リダクション $R(G,x\rightarrow y)$ を施して頂点 $x,y\in V$ と辺 $x\rightarrow y$ を頂点 x に縮退するまでは，いずれのグラフにおいても $x\ \mathbf{idom}\ y$ は成立し不変である．また，遅くともリダクション $R(G,x\rightarrow y)$ を施す時点までには $P(y)=\{x\}$ が成り立つ．□

5. 計 算 量

グラフの頂点数を N，辺数を M として，本算法の計算量を検討する．

（1）初期化部分　図 6 の行 13 による 3 件の初期化はいずれも計算量 $O(N)$ であり，行 14〜18 の初期化はすべての辺を 1 度ずつアクセスするので $O(M)$ である．

（2）頂点依存部分　リダクションループの本体において，先頭の行 21，行 22 はリダクション対象頂点について，各 1 回実行される．リダクションは入口頂点以外の頂点について，必ず 1 回に限り発生するので，全リダクションでのこの部分の計算量は $O(N-1)$ である．

（3）辺依存部分（前半） 行 23〜29 では，対象–先行頂点 x を始点とする辺がアクセスを受ける．対象–先行頂点としての最初の x アクセス時点にフラグテーブルを作成し，保存して使用すれば，その後のリダクションによって発生するフラグテーブルの修正は $O(1)$ なので，後続頂点の直和計算の中に含めることができる．したがって，全リダクションでのこの部分の計算量は，各頂点の初期状態で見積もればよいので，$O(M)$ である．

すべての頂点についてそれぞれ固有のフラグテーブルを用意すれば，この議論は正しい．本稿では 1 つのフラグテーブルを共用したので，グラフの構造によっては，再設定が起きる，しかしこれによる計算量の増加は各頂点の後続頂点の数に依存するので，通常の制御フローグラフでは N および M に無関係な定数と見なせる．

（4）辺依存部分（後半） 行 30〜39 では，対象頂点 y を始点とする辺がアクセスを受ける．これについては，前項（3）のように情報を保存して利用することは難しい．しかしこれらの辺の終点は，y の支配辺境の要素であって，その個数は 0 または 1 の場合が多い．複数の場合もありうるが，次章に示す実例では，GOTO を多用したサンプルで最大 4 個であった．その個数は後続頂点の数に依存し，N や M とは無関係である．対象頂点 y となる頂点の個数は $N-1$ であり，各頂点のこの部分の計算量は $O(1)$ である．したがって，全リダクションでのこの部分の計算量は $O(N-1)$ である．

計算量は $O(N)$ と $O(M)$ の和であるが，分岐のないグラフで $N-1=M$，分岐を 1 つでも含む場合は $N \leq M$ なので，結局，全体の計算量は $O(M)$ である．

6. 実例による確認

支配木・支配辺境の検出動作の確認と計算量の計測を行うため，本算法を実装し他の算法との比較を行った．以下本算法によるプログラムを**CODE-S** と呼ぶ．他の算法としては Ochranova による算法[5] を用いた．Ochranova の算法は，プログラムコードでは示されていないので，提示された算法に沿ってコード化した．これを**CODE-O** と呼ぶ．Ochranova の方法は，本算法と同様にリダクションに基づく方法であり，その概要は以下のとおりである．

グラフの入口頂点を探索頂点 x として，繰返しを開始する．繰返しの判定 L は，x の後続頂点の中に x を唯一の先行頂点とする頂点があれば☆，それを候補頂点 y として繰返しの本体を実行し，なければ繰返しを終了する．繰返しの本体では，まず y を支配木に入れる．次に y が支配頂点でなければ（後続頂点がないか，後続頂点があっても，そのすべてが複数の先行頂点を持っていれば）☆☆①を，そうでなければ②を選ぶ．①は，y を x にリダクションする手続き R を実行し，その結果 x が支配頂点でなくなっていれば☆☆☆，x の先行頂点を次の探索頂点 x とし，そうでなければ，探索頂点 x を変えずに L に戻る．②は，y を新たな探索頂点 x として L に戻る．y を x にリダクションする手続き R は，x の後続頂点集合から y を除去し，y の後続頂点集合を x の後続頂点集合に合併（直和）する．次に y の各後続頂点の先行頂点集合から y を除去し，x がなければ x を追加する．

以上が Ochranova の算法の概要である，本稿の方式と異なりスタックを使わず，脚注に示すようにたびたび先行頂点や後続頂点のアクセスと判定を必要としている．これらの操作はスタックを利用する本稿の方式ではすべて不要である．また，先行頂点については，唯一の先行頂点番号のみが必要であり，スタックに積む時点で明らかになるので，本稿の方式では先行頂点の集合は必要ではない．本稿の方式は，集合の操作が後続頂点のリダクションに限られ，計算が必要最小限に抑えられるとともに，計算量の見通しがつけやすい簡潔な制御構造を与え，実装と保守の容易なアルゴリズムを提供する．

CODE-O のコード化については，CODE-S との比較において，実装の差異による計測差が出ないように次の配慮を行い，できるだけ公平を期した．

☆ スタックを使う方式に比べ，後続頂点の探索が必要である．
☆☆ 同上．
☆☆☆ 同上．

表 1 計算量の統計値
Table 1 Statistics of complexity.

	N	M	McO/M	McS/M
maximum	346	454	7.78	2.50
average	32.29	40.71	4.05	1.46
std.dev	44.29	56.63	1.05	0.32

N : Number of vertices
M : Number of edges
McO : Complexity of CODE-O
McS : Complexity of CODE-S

- 後続頂点の集合は同じリスト構造にする．
- 先行頂点集合の直和計算は計測から除外する☆．
- 後続頂点集合の直和計算は同じ方式を用いる．

計測に使った言語処理系は Microsoft VC++ version 6.0 である．また，計算量は頂点依存部分については明白なので除外し，辺依存部分の実行回数を調査した．実例データは，SpecCFP92[11] の FORTRAN77 によるソース 240 本余りから，非可約なループを含む 1 本を除いて，前処理により 241 個の制御フローグラフを取り出し，サンプルとして使用した．前処理プログラムの機能は，FORTRAN77 の言語特性と，制御フロー認識という限定目的から，単純に言語のキーワードによる解析を行った．対象としたプログラムは，GOTO 命令を多数含む例も，計算量の多い制御フローに遭遇する可能性が高くなるとみて，あえて選んだ．このため，得られたグラフ構造は，単純なものから複雑なものまで変化に富み，全体として，実用レベルであるが若干複雑といった性格を備えている．

計算量の計測は，支配関係解析の実行部で，辺数（M）への依存部分での辺の参照回数をカウントして，それぞれの計算量 Mc とした．

以上により，同じサンプルの集合を 2 種類の支配関係解析プログラムで処理した．サンプルごとの N，M と，Mc の M に対する比 Mc/M についての統計値を**表 1** に示す．両者を比較すると，Mc/M が，CODE-O に比べて CODE-S では大きく減少している．また，いずれの場合も，$Mc = kM$ とすれば，k は 1 桁台である．しかし，Mc が M に比例し $O(M)$ であるかどうかについては，M に対する Mc の分布を見る必要がある．

図 7，**図 8** は，サンプルごとの M を横座標，Mc を縦座標とした散布図である．ここでは，分布の傾向を見るのが目的なので，プリント出力により概略を示した．そのため分布の密な領域では点が重なっていて，

☆ Ochranova の算法では先行頂点集合を使っているが，本算法では先行頂点数と先行頂点番号（1 個分）のみ用い，先行頂点集合は使っていない．

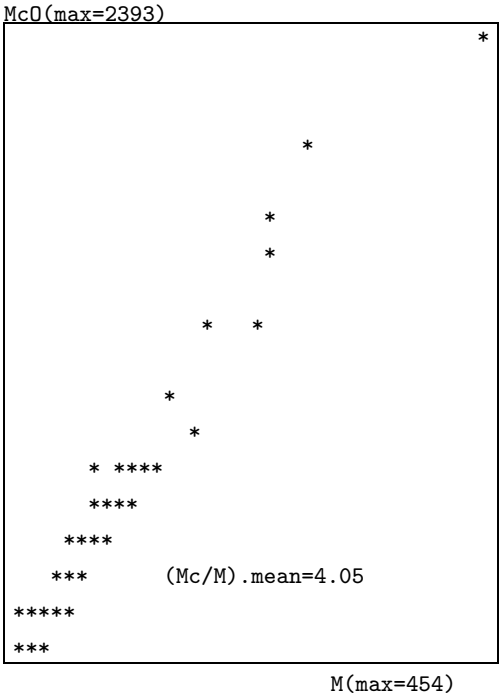

図 7 散布図：CODE-O
Fig. 7 Scatter diagram: CODE-O.

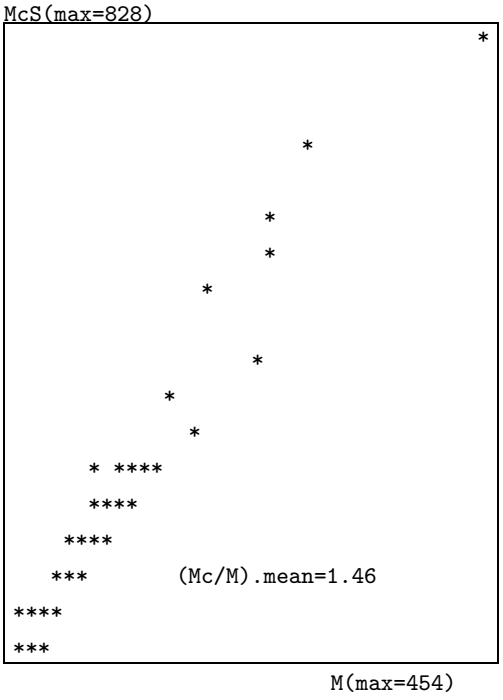

図 8 散布図：CODE-S
Fig. 8 Scatter diagram: CODE-S.

重みが隠れているが，Mc が $O(M)$ であるかどうかを観察するためのグラフとしては，原点を通る直線に近い分布を示しているこの図の精度で十分である☆．

両散布図における点の分布は，誤差をともなう直線的関数関係のある 2 変量の相関図として妥当な状況を示している．ただし，ここでの回帰直線からのばらつきは，狭い意味での誤差ではなく，フローグラフの構造の差に基づく偏差であり，M 以外の要因から発生して計算量に影響している．この変動はグラフの頂点数や辺数といった大きさには直接関係せず，フローグラフの形に起因するものと考えられる．散布図が示している直線的関数関係は，対象としたサンプルの，実用プログラムに共通する特性を統計的に示しているといえる．

Mc が $O(M)$ であるとすれば，表 1 の示す Mc/M の値（$Mc = kM$ の k の値）の比は 3:1 程度である．これは 2 種類の解析プログラムの能率の違いを示すものであり，CODE-S において，スタックを利用してリダクション候補頂点を毎回探索する操作を無用化したことの効果が大きいと考えられる．

7. おわりに

多くのプログラムは可約な制御フローグラフ構造を持っている．制御フローグラフの支配木や支配辺境を求めるような基本的な算法は，計算量が少ないことのほかに，できるだけ実装が容易で保守性に優れていることが望ましい．

プログラムの実装と保守を容易にするには，算法が単純で整理されていることが有用である．可約なグラフは，適切な順に選択した頂点のリダクションを逐次実行して，オリジナルのグラフを入口の頂点 1 つに変換縮小できる．この過程で，頂点をリダクションするたびに，その頂点を支配木に追加でき，リダクションの時点での頂点の直接後続頂点の集合は，以後リダクションの終了までそのまま保存され，結果的に各頂点の支配辺境を与える．

本算法では，リダクション対象頂点の選択方法を研究し，対象頂点の候補の番号をスタックに並べ，これを更新しつつ使用することで，リダクション進行中に頂点の集合から対象頂点を探索する操作を不要にした．この算法の計算量は $O(M)$ である．これにより，リダクションループの本体について簡潔な記述を行い，実装容易な算法が得られた．

算法をプログラム化し，SpecCFP92 の FORTRAN77 プログラム，約 240 件から取得した制御フローをサンプルとして，計算量を計測した．その結果は実用プログラムの場合は計算量 $O(M)$ であるとする主張を補強するものであった．リダクションに基づく別算法[5] に従ったプログラムも，計算量 $O(M)$ を裏付ける結果を示しているが，係数としては約 3 倍の計算量であった．$Mc = kM$ の係数 k の値で 3 倍なので，プログラムのサイズ M が大きい場合の計算量の差は無視できない．

☆ 表 1 では，この重みは考慮されている．

謝辞 本研究に関して電気通信大学情報工学科渡邊坦研究室の樽石将人君に積極的な議論をいただいた．また，本稿の査読者の方には周到で有益なご意見をいただいた．ここに記して深く感謝の意を表する．

参考文献

1) Lowry, E. and Medlock, C.: Object code optimization, *Comm. ACM*, Vol.12, No.1, pp.13–22 (1969).
2) Aho, A.V. and Ullman, J.D.: The Theory of Parsing, *Translation, and Computing, Vol.II: Compiling*, Prentice-Hall, Englewood Cliffs. N.J., (1972).
3) Aho, A.V., Hopcroft, J.E. and Ullman, J.D.: On finding the least common ancestors in trees, *ACM Symposium on Theory of Computing*, Austin, Texas (1973).
4) Tarjan, R.E.: Finding Dominators in Directed Graphs, *SIAM J. Comput.*, Vol.3, No.1, pp.62–89 (1974).
5) Ochranova, R.: Finding dominators, *Proc. FCT'83 Borgholm Sweden*, LNCS 158 Foundaton of Computation Theory, pp.328–334 (1983).
6) Harel, D. and Tarjan, R.E.: Fast Algorithms for Finding Nearest Common Ancestors, *SIAM J. Comput.*, Vol.13, No.2, pp.338–355 (May 1984).
7) Aho, A.V., Sethi, R. and Ullman, J.D.: *Compilers Principles, Techniques and Tools*, Addison-Wesley, Reading, Massachusetts (1986).
8) Gabow, H.N.: Data structure for weighted matching and nearest common ancestors with linking, *Proc. 1st Annual ACM-SIAM Symposium on Discrete Algorithms*, pp.434–443 (Jan. 1990).
9) Aho, A.V., Hopcroft, J.E. and Ullman, J.D., 野崎昭弘，野下浩平（共訳）：アルゴリズムの設計と解析，サイエンス社 (1993).
10) Ramalingam, G. and Reps, T.: An incremental algorithm for maintaining the dominator tree of a reducible flowgraph, *Conference*

Record of the 21st ACM Symposium on Principles of Programming Languages, pp.287–298 (Jan. 1994).
11) Giladi, R. and Ahituv, N.: SPEC as a Performance Evaluation Measure, *IEEE Computer*, Vol.28, No.8, pp.33–42 (1995).
12) Havlak, P.: Nesting of Reducible and Irreducible Loops. *ACM Trans. Prog. Lang. Syst.*, Vol.19, No.4, pp.557–567 (1997).
13) Alstrup, S., Harel, D., Lauridsen, P.W. and Thorup, M.: Dominators in linear time, *SIAM J. Comput.*, Vol.28, No.6, pp.2117–2132 (1999).
14) 中田育男：コンパイラの構成と最適化，朝倉書店 (1999).
15) Ramalingam, G.: On loops, dominators and dominance frontiers, *Proc. SIGPLAN '00 Conf. on Programming Language Design and Implementation*, pp.233–241 (Jun. 2000).

(平成 13 年 5 月 31 日受付)
(平成 13 年 7 月 31 日採録)

齋藤　鐵男（学生会員）

昭和 5 年生．昭和 28 年東京大学工学部電気工学科卒業．同年三菱鉱業（株）入社．技術計算，OR 計算プログラム開発に従事．昭和 42 年東京芝浦電気（株），昭和 45 年より東京技術計算コンサルタント（株）．平成 2 年東京大学工学部工学系大学院研究生修了．現在東京電機大学工学部非常勤講師，電気通信大学大学院電気通信学研究科（博士後期課程）情報工学専攻．電気学会会員．

鈴木　　貢（正会員）

電気通信大学情報工学科助手．記憶管理アルゴリズム，並列/分散アルゴリズム，言語処理系等に興味を持つ．ACM，電子情報通信学会，日本ソフトウェア科学会各会員．

渡邊　　坦（正会員）

昭和 37 年京都大学理学部数学科卒業．日本 IBM（株），（株）日立製作所中央研究所，同システム開発研究所を経て，平成 6 年より電気通信大学情報工学科教授．工学博士．プログラミング言語とプログラミング・ツール，各種言語処理系の開発を行ってきた．現在は，コンパイラの研究・開発を主要テーマとしている．著書「コンパイラの仕組み」（朝倉書店）ほか．日本ソフトウェア科学会，ACM，IEEE 各会員．