非可約な制御フローグラフのための簡潔で高速な支配木と支配辺境の検出算法

齋 藤 鐵 男† 鈴 木 貢†† 渡 邊 坦††

非可約な制御フローグラフとは複数の入口を持つ非可約なループを含む制御フローグラフをいう．本論文ではそのようなグラフの支配木と支配辺境を求める簡潔で高速な算法を報告する．非可約なループは，高級言語ではあまり現れないが，コンパイラの最適化の操作により発生する可能性がある．本算法は，グラフの支配関係を保存する変換を施した DAG を生成し，その縮約とその後の逆変換により，非可約なグラフの支配木と支配辺境を求める．通常の可約なグラフも同じ算法で計算できるので，単一の算法ですべての制御フローグラフに対応が可能である．本算法は，深さ優先探索，強連結成分の検出，DAG の縮約のみを基本としており，実現にあたって複雑で特殊なデータ構造は必要としない．計算量はグラフの辺の数を M として $\mathrm{O}(M)$ である．本算法を C 言語で実装し，パソコン（500 MHz Pentium II）で実行した処理速度は，SPEC CPU2000 ベンチマークプログラムから得た約 1,700 件のグラフに対して，グラフの辺あたり平均で約 2.4 μs であった．

A Concise and Fast Algorithm for Irreducible Control Flow Graphs to Identify Immediate Dominators and Dominance Frontiers

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

The irreducible control flow graph is a control flow graph with irreducible loops that has multiple entries. We present a concise and fast algorithm which can identify immediate dominators and dominance frontiers in such graphs. The irreducible loops are rare in programs written in high-level programming languages, but they may be brought in by optimizations of compilers. This algorithm uses a kind of DAG derived from the original graph with some modification to preserve dominace relations of the graph, then reduces it and gets the dominator tree and dominance frontiers of the original graph by the final reverse modification. As this algorithm is valid too for ordinary reducible graphs, it is effective for all kind of control flow graphs. It uses only “depth first search”, “strongly connected components identification”, and “reduction of DAG”, so there are no need for any understanding of complex and special data organizations. The complexity of the algorithm is $\mathrm{O}(M)$, where M is the number of edges. The processing speed of the algorithm implemented in C, on a personal computer (500 MHz Pentium II) showed 2.4 μs/edge in average for about 1,700 graphs derived from SPEC CPU2000 benchmark programs.

1. はじめに

本論文は，任意の制御フローグラフ（CFG，control flow graph）の支配木と支配辺境を，縮約によって求める算法を述べる．本算法は，CFG が可約であるか非可約であるかを問わず同じ手続きで扱い，理解と実装が容易である．本算法を C 言語で実現し，CFG 約 1,700 件を例題として実測した結果，計算量は辺の数を M として $\mathrm{O}(M)$，処理速度は有向辺 1 辺あたり約 2.4 μs であった．

CFG は，プログラムの制御の流れを表すグラフである．それは基本ブロックを頂点とし，頂点間の有向辺で制御の流れを示す連結な有向グラフであり，ただ 1 つの入口を持つ．CFG の繰返しループは，入口が 1 つならば可約なループ，入口が複数ならば非可約なループと呼ばれる．また，非可約なループを含む CFG は非可約な CFG（irreducible control flow graph），非可約なループを含まない CFG は可約な CFG（reducible control flow graph）といわれる．非可約な CFG はループの検出に手数がかかるほか，検出手法によって

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

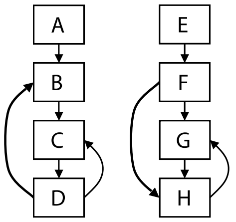

図 1 制御フローグラフ
Fig. 1 Control flow graph.

はループの可約，非可約の認識が一意に定まらないことがある[8] 等，扱いにくい性質がある．**図 1** (a) に可約な CFG, 図 1 (b) に非可約な CFG の例を示す．図 1 (b) では G，H が 1 つのループの 2 つの入口である．非可約な CFG の支配木を求める発表は多数あるが，非可約であるとコンパイラでのある種の最適化をあきらめることも言及されている[8]．

Fortran77 以後の構造化言語で実装されたプログラムでは非可約な CFG の発生する可能性が低い．また構造化言語以前でも，ほとんどのプログラムは可約なグラフ構造になっているという意見[3] もある．

一方，コンパイラの最適化処理の中で非可約な CFG が発生する機会は否定できない．このため，非可約な CFG も例外とせず，CFG の支配木と支配辺境を検出する研究は有用である．

本論文では，非可約な CFG と可約な CFG を同じ手続きで扱う簡潔な算法を提示する．本算法は DAG の縮約により支配木と支配辺境の検出を同時に行い，算法固有の特殊なデータ構造は必要とせず，理解と実装が容易である．縮約は CFG から導出した DAG 構造に基づいて行うが，特定の場合に限り部分グラフを追加して DAG 化による支配関係の変化を防ぐ．縮約によって得られる支配木と支配辺境は，これらの補助的な情報を含むが，簡単に削除してもとの CFG の支配木と支配辺境をうることができる．本算法を C 言語 600 行余りで実現し，SPEC CPU2000 のプログラムをコンパイルした結果のアセンブリリストから得た CFG 約 1,700 件を例題として，算法の計算量と時間を実測した．実用上はありえないような非可約な CFG でも正しい結果を与えるのはもちろん，計算量は辺の数を M として $O(M)$ であることの十分な傍証を得た．

2. 関 連 研 究

1969 年，Lowry ら[9] は OS/360 Fortran H コンパイラの最適化処理に関して，共通部分式の削除や，レジスタ割付け等の話題とともに，CFG と直接支配頂点の役割を述べている．これが支配関係に関する最初の言及といわれており[5]，以後，支配木やループに関する研究が相次いで発表されることになる．1972 年に Aho ら[4] は頂点数 N，辺数 M に対して $O(N(N+M))$ 時間の算法を提案し，翌 1973 年，Aho ら[1] は，可約な CFG に対して，木構造の中で最近共通祖先を求める方法で，$O(N + M\,logM)$ 時間の算法を示した．さらに 1974 年，Tarjan[17] が効率的な集合和の計算を用い，深さ優先探索の情報に基づいて任意の CFG の支配木を計算する $O(N\,logN + M)$ 時間の算法を発表した．また，Tarjan[16] は多くの最適化の問題が可約な CFG を対象に解決されてきたとして，1974 年に，CFG が可約であることを判定するアルゴリズムを提案した．これは後に Havlak[8] の算法の基礎となった．

可約な CFG について，Ochranova[10] は縮約操作で支配木を求める算法を提案し，計算量は $O(M)$ であると主張している（1983）．一方，最近共通祖先の算法については，Harel ら[7] が完全 2 分木に基づいて 2 頂点の最近共通祖先が $O(1)$ で求めうることを利用して，$O(N+M)$ の技巧的な算法を示した（1984）．しかし，この方法は複雑なデータ構造を用いており，縮約操作による算法のような簡潔性は望めそうにない．

1990 年代には，Steensgaard[15] が強連結成分の検出に基づいて，ループのネストを見出すことを提案した（1993）．Ramalingam ら[13] は，CFG の辺の増減にともなう支配木の部分的な修正算法を論じ，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ことを述べている（1994）．Havlak[8] は Tarjan[16] の可約性判定アルゴリズムを基礎に，任意の制御フローから，ループのネストの木を検出する算法を示した（1997）．Alstrup ら[5] は一般の CFG について，Tarjan の算法を改善した方法を提示した（1999）．その中で論文 10) 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 CFG の支配木を求める $O(N+M)$ 時間の算法を示した．Ramalingam[11] は Havlak[8]，Sreedhar ら[14]，Steensgaard[15] の算法の比較と計算量削減について述べ（1999），さらに，一般の CFG について，抽象化したループ検出のフレームを提案し，支配木の検出のために非可約な CFG の逆辺を除いて DAG 化する場合に，失われる情報を補償する手段として仮想頂点の挿入に言及した[12]（2000）．ただし，その支配関係への影響についての十分な証明はなく，DAG 化後の支配木の検出については，Gabow[6]（1990）のデータ構造を用いた最近共通祖先の計算に

より，$O(N)$ で計算できるとだけ述べている．

なお，DAG の支配木を検出する方法については，Ochranova[10] が縮約による算法を発表したほかは，最近共通祖先の算法自体の発表や，最近共通祖先によると述べた論文が多い．しかし，これまでの研究では，最近共通祖先による支配木の検出は縮約による方法よりも算法が複雑であり，支配辺境については別途計算を要する．また，Ochranova[10] の縮約による算法では，原理的に同時に支配辺境が求められるはずであるが，それについては何もふれていない．

本論文では，齋藤，鈴木，渡邊[19] の，DAG の支配木と支配辺境を縮約によって求める算法を一般の CFG 用に拡張する方法を研究し，DAG では扱わなかった逆辺（back edge）の，縮約における取扱いを考案した．また，非可約なループのうち，プリヘッダ（後述）が複数の場合に限り仮想頂点を導入して，可約な CFG と同様に支配木と支配辺境を求める方法を導入し，それについて，正しさの証明を与え，実装して確認および計算量の観測を行った．

3. 用　　語

ここでは本算法で用いる基本的な概念について述べる．ここで扱う CFG では，プログラムの基本ブロックまたはその集合を頂点とし，制御の流れを有向辺で表す．頂点の集合を V，有向辺の集合を E，プログラムの入口のブロックを s として，CFG G を $G=(V,E,s)$ で表す．この形式は，グラフ $G=(V,E)$ における入口 s を明示するための表現である．

CFG の頂点は通常 1 個の基本ブロックであるが，本算法では CFG の頂点を基本ブロックの集合と考え，頂点を代表する基本ブロックが後述の規則により代表ブロックとして定まるものとする．単一の基本ブロックからなる頂点ではその基本ブロックが代表ブロックである．一般的には頂点は基本ブロック $x, x_1, x_2, \ldots, x_n$ の集合であり，x が代表ブロックのとき，$\{x, x_1, x_2, \ldots, x_n\}$ と，x を先頭に表示するか，またはその基本ブロックの集合を $v(x)$ と表示する．1 個の基本ブロック x からなる頂点は $v(x)=\{x\}$ である．ただし，文脈上明らかなときは，$v(x)$ の代わりに x を用いる．

頂点 $v(y)$ を頂点 $v(x)$ に統合し，統合後の代表ブロックを規則に従って選び直すことを，頂点の併合といい，$v(x) \equiv v(x) \cup v(y)$ または $v(x) \cup v(y)$ と書く．併合は後述する縮約に際して行われる．このときの $v(y)$ を併合頂点，$v(x)$ を併合先頂点という．併合した結果の頂点は,「併合先頂点の代表ブロックを用いる」という規則に従って代表ブロックを定める，たとえば $v(x)=\{x\}$，$v(y)=\{y\}$ であるとき，$v(x) \equiv v(x) \cup v(y) = \{x, y\}$ である．この $v(x)$ を，$v(u)=\{u\}$ に併合すると $v(u) \equiv v(u) \cup v(x) = \{u\} \cup \{x, y\} = \{u, x, y\}$ となる．縮約における，この状況を**図 2** に示す．

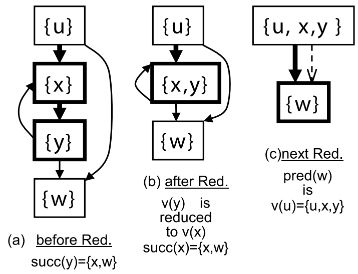

図 2　頂点の縮約
Fig. 2　Reduction of vertices.

頂点 $v(x)$ から $v(y)$ に至る有向辺を $x \rightarrow y$ と表示し，以下では単に辺と呼ぶ．辺 $x \rightarrow y$ の x を辺の始点，y を終点という．辺の始点・終点を端点ともいう．辺の端点には代表ブロックを用い，代表以外のブロックは頂点の要素であっても，辺の端点とはしない．頂点 v(x) と頂点 v(y) の対に対しては，辺 $x \rightarrow y$ と辺 $y \rightarrow x$ がそれぞれたかだか 1 つ存在しうる．x の後続頂点（successor）の集合を $succ(x)$ で表し，y の先行頂点（predecessor）の集合を $pred(y)$ で表す．頂点の縮約とは，辺 $x \rightarrow y$ があるとき，代表ブロック y の入出辺を x の入出辺におきかえるとともに，頂点 $v(y)$ を頂点 $v(x)$ に併合し，$v(x) \equiv v(x) \cup v(y)$ とすることをいう．すなわち $u \rightarrow y$，または $y \rightarrow w$ があれば，それぞれ $u \rightarrow x$，$x \rightarrow w$ におきかえ，$y \rightarrow x$ は $x \rightarrow x$ とする．ただし $x \rightarrow y$，$y \rightarrow y$ は除去する．以後，頂点 $v(y)$ を頂点 $v(x)$ に縮約することを，$v(x) \equiv v(x) \Leftarrow v(y)$ と表す．図 2 に頂点の縮約の前後の様子を示す．

頂点 $v(u) \in V$ から頂点 $v(w) \in V$ に至る経路（path）$u \Rightarrow w$ とは，辺の並び $u \rightarrow u_1$，$u_1 \rightarrow u_2$，$u_2 \rightarrow \ldots \rightarrow u_k$，$u_k \rightarrow w$ をいう．経路の途中に頂点 $v(p)$ があることを示す必要があるときは $u \Rightarrow p \Rightarrow w$ と表す．また，経路の途中の辺を明示するときは，$u \Rightarrow p \rightarrow q \Rightarrow w$ のように表す．$v(u)$ から $v(w)$ までの経路が存在するとき，$v(w)$ は $v(u)$ から到達可能であるという．$G=(V,E,s)$ では入口 $v(s)$ からすべての $v(w) \in V$ に到達可能であるとする．また，すべての頂点は自分自身に到達可能であると見なす．経路 $u \Rightarrow u$ が 1 つ以上の辺を含み，経路上の u 以外の頂点に重複がないとき，これをサイクルという．サイクルのないグラフは非循環グラフ（DAG: Directed Acyclic Graph）

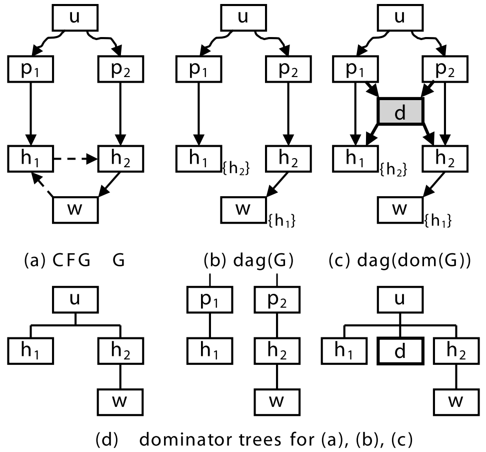

図 3　非可約ループへの仮想頂点 v(d) の追加
Fig. 3　Add dummy vertex v(d) to irreducible loop.

と呼ばれる．$v(s)$ から $v(w) \in V$ へのすべての経路上に存在する頂点 $v(u) \neq v(w)$ を頂点 $v(w)$ の祖先という．$v(w_1), v(w_2), \ldots, v(w_k) \in V$ のそれぞれの祖先に共通で，最も $v(s)$ から遠い頂点を $v(w_1), \ldots, v(w_k)$ の最近共通祖先（nca： nearest common ancestor）と呼び，$nca(w_1, \ldots, w_k)$，$nca(w_i, 1 \leq i \leq k)$ 等で表す．**図 3** は，非可約なループの DAG 化において仮想頂点の追加が必要な場合についての説明図である（この場合についての説明は 4.6 節「仮想頂点の挿入」を参照）．DAG 化とは，グラフの中でループヘッダに戻る逆辺を無視して，グラフの中の DAG 構造に注目することをいう．同図の (b)，(c) において，頂点 h_1，w の右にそれぞれ $\{h_2\}$，$\{h_1\}$ がついているのは，CFG としては $h_1 \rightarrow h_2$，$w \rightarrow h_1$ が存在するが，DAG 構造としては $h_1 \rightarrow h_2$，$w \rightarrow h_1$ を無視する（逆辺だから）ので，有向辺の矢印に代えて後続頂点の集合として添付して示したものである．同図の (a)，(b)，(c) いずれにおいても，p_1，p_2 の nca は u である．

グラフ $G = (V, E)$ の部分グラフ $G_S = (V_S, E_S)$，$V_S \subseteq V$，$E_S \subseteq E$ において，任意の $v(u)$，$v(w) \in V_S$ が相互に到達可能であるとき，グラフ G_S はグラフ G の強連結成分（SCC: Strongly Connected Component）であるという．単一の頂点 $v(u)$ は自己ループをなすサイクル $u \rightarrow u$ を持つ場合に限り SCC と見なす．SCC は $v(s) \in V_S$ であるか，そうでなければ，少なくとも 1 つの辺 $p \rightarrow h$，$v(p) \in (V - V_S)$，$v(h) \in V_S$ を有する．上記の $v(s)$ または $v(h)$ をヘッダ（header），$v(p)$ をプリヘッダ（preheader）という．また，頂点 $v(w) \in V_S$ からヘッダへの辺を逆辺（back edge）という（**図 4**）．

SCC G_S の逆辺のみを取り除いたグラフの中に，さ

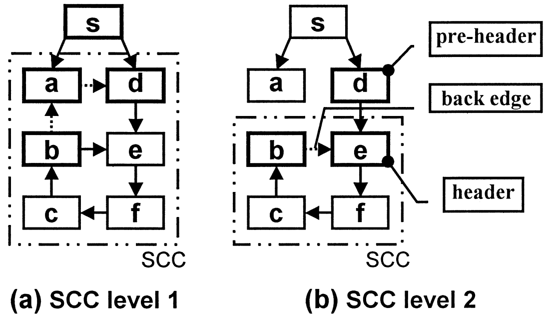

図 4　SCC，ヘッダ，プリヘッダ，逆辺
Fig. 4　SCC, header, pre-header, back edge.

らに SCC $G_{SS} \subset G_S$ が存在するとき，G_{SS} をグラフ G_S の入れ子の SCC という．G の中に互いに入れ子でない SCC G_1，G_2 が存在すれば，$G_1 \cap G_2 = \phi$ である．$G1$，$G2$ の中に入れ子の SCC がそれぞれ複数あれば，その間に同様の関係があるから，これらすべての SCC は G を根とし，各 SCC を節点とする木で表すことができる．この木の節点の深さを SCC のレベルまたは SCC の入れ子の深さという．

SCC の頂点に相互の到達可能性を与えているものは SCC の中にあるサイクルであり，サイクルはループとも呼ばれる．SCC のヘッダ，プリヘッダはループのヘッダ，プリヘッダでもある．ヘッダが 1 つしかないループは可約なループ（reducible loop）と呼ばれ，複数のヘッダを持つループは非可約なループ（irreducible loop）と呼ばれる．非可約なループを含まないグラフは可約なグラフ（reducible graph），非可約なループを含むグラフは非可約なグラフ（irreducible graph）と呼ばれる．

$G = (V, E, s)$ では，一般に $v(s)$ から $v(w) \neq v(s)$ への経路 $s \Rightarrow w$ は複数存在しうる．そのいずれにも共通に存在する頂点 $v(u)$ を $v(w)$ の支配頂点（dominator）といい，$u\ dom\ w$ または $u \in dom(w)$ と書く．任意の頂点 $v(u)$ は $u\ dom\ u$ である．頂点 $v(u)$ に対して，集合 $\{w \mid u\ dom\ w\}$ を頂点 $v(u)$ の支配域といい，$DR(u)$ で表す．$v(u) \neq v(w)$ で，最も $v(w)$ に近い $v(w)$ の支配頂点 $v(u)$ を $v(w)$ の直接支配頂点（immediate dominator）という．このとき，$v(u)$ は $v(w)$ を直接支配するといい，$u\ idom\ w$，または $u = idom(w)$ と書く．図 3 (a) では $idom(h_1) = idom(h_2) = u$，図 3 (b) の DAG 構造では $idom(h_1) = p_1$，$idom(h_2) = p_2$，図 3 (c) の DAG 構造では，$idom(d) = idom(h_1) = idom(h_2) = u$ である．$G = (V, E, s)$ において，頂点 $v(w) \in V - v(s)$ は，ただ 1 つの直接支配頂点を持つ．したがってすべ

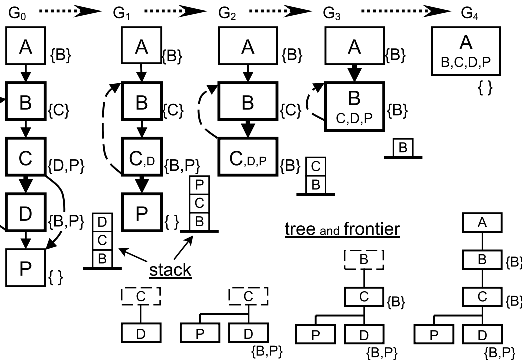

図 5 可約な CFG の縮約過程
Fig. 5 Reductions for reducible CFG.

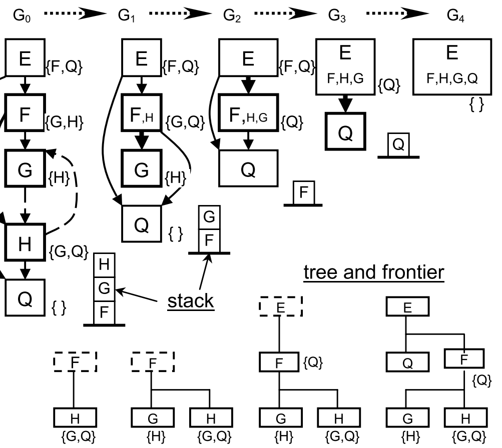

図 6 非可約な CFG の縮約過程
Fig. 6 Reductions for irreducible CFG.

ての $v(w) \in V - v(s)$ について $u = idom(w)$ を親の節点，w を子の節点として，s を根とする木を作ることができる．この木を支配木（dominator tree）という（図 3 の (d)，**図 5**，**図 6** の tree）．木の根以外の任意の節点 w に対して，その親の節点 u を $u = parent(w)$ と書く，節点 w が節点 u の子の節点の 1 つであるとき $w \in child(u)$ と表す．支配木 T がグラフ G の支配木であることを示すには $T(G)$ と書く．w が T の節点であることを $w \in T$ と表す．

頂点 $v(u) \in V$ の支配辺境（dominance frontier）とは，$v(u)$ から有向辺の方向にたどって，初めて $v(u)$ の支配を脱する頂点の集合であり[18]，これを $DF(u)$ と書く（図 5，図 6 の frontiers）．頂点の集合 V があるとき，V の全頂点の代表ブロックの集合 $\{w \,|\, v(w) \in V\}$ を，$rep(V)$ で表す．$succ(u) = \phi$ ならば $DF(u) = \phi$ である．$succ(u) \neq \phi$ ならば $DF(u) = \{q \,|\, p \rightarrow q \in E, p \in DR(u), q \notin DR(u)\}$ である．

4. 算　　法

図 7 に任意の CFG に対する支配木と支配辺境を求める算法を手続き *DOMINANCE* として示し．以下に概要を説明する．文中では図 7 の行番号 nn を Lnn で示す．

4.1 主なデータ

入力は一般の CFG $G = (V, E, s)$ であるが，ここでは，表現 $G(V, S, s)$ を算法の入力とする．V は頂点データの配列，S は頂点ごとの後続頂点リストの集合，s は CFG 入口頂点の指定である．すなわち，v を始点とする辺の終点の頂点番号の集合を $S(v)$ で表し，すべての $v \in V$ に対する $S(v)$ の集合を S と記す．出力については，頂点 $v \in V$ に対して直接支配頂点の番号を $idom[v]$ に格納して支配木情報とする．支配辺境は，手続き *DOMINANCE* の進行にともない部分集合 $S(v)$ に保存される．変数 x，y，v は頂点の番号で，v は一般の頂点，y は縮約される頂点，x は縮約で y の縮約先となる頂点の番号である．$level$ は SCC の入れ子の深さを示す．配列 $Stack[\,]$ は単 1 の先行頂点を持つ頂点（以下では単先行頂点という）の番号を，深さ優先探索の後付け番号順に保存し，逆順にとりだすスタックである．また $Npred[\,]$ は先行頂点数のカウンタで，CFG の縮約に応じて更新することで，縮約の過程で単先行頂点となった頂点を識別する．

4.2 図 5，図 6 に共通な事項

以下，本算法を例を用いて説明する．厳密な論理は次章を参照されたい．図 5，図 6 の上端に並ぶ記号 G_n は，各縮約段階のグラフの標識で，左端がオリジナルのグラフである．これらのグラフとは別に，支配木（tree）と各頂点の支配辺境（frontier）を並べて示す．グラフの頂点は大きい矩形の中に代表ブロックを太字で表した．矢線は有向辺で，実線は DAG の辺，点線は逆辺である．太い矢線はそのグラフで縮約する辺を表し，その終点を始点に縮約する．縮約の結果，1 つ右のグラフに移る．また，太い線の矩形は代表頂点番号がスタックに積まれていることを示す．頂点の矩形のそばにある集合表示 { } は，後続頂点の集合であり，文字のない場合は空集合である．支配木の節点に添記した後続頂点の集合は，その頂点を縮約したときの後続頂点に等しく，その頂点の支配辺境を表す．本算法では，縮約される頂点 y は，単先行頂点で，かつ $S(y)$ が空であるか，空でなければ $S(y)$ の要素すべてが y の支配を受けない状況になっている．したがって，縮約を受ける時点の $S(y)$ は y の支配辺境に

```
01 PROCEDURE DOMINANCE     /*  Identify Dominator Tree and Dominance Frontiers. */
02 INPUT   G=(V, S, s)       //  General CFG, V is the set of vertices, N=|V|,
03                           //   S(v) is the set of Successors for v in V C
04 OUTPUT  idom[v]           //  Immediate Dominator for vertex v in V.
05 BEGIN PROC
06 // (1) Prepare DAG for the graph G, with dominance relations unchanged.
07     Clear flags of all vertices and edges in G.
08     level = 0
09     DO  level = level+1
10         Number all v in V by the pre-order DFSN
11         Identify SCCs in G, ignoring back edges flagged in preceeding transformation.
12         Set flag for headers, preheaders and back edges of new SCC found.
13         Add Dummy Vertex and Dummy Edges for every SCC, if necessary.
14     WHILE new SCC is found.
15 // (2) Initialize for Reduction.        /* Prepare stack of candidates for reduction.*/
16     Clear idom[v], Npred[v] for all v in V. Assign the post-order DFSN to all v.
17     Clear Stack.
18     FOR ALL y in V, ascending post-order DFSN DO
19         FOR ALL sy in S(y) DO
20             IF edge y to sy is not a back edge THEN // Select edge of DAG.
21                 idom[sy] := y                 // Store y into idom[sy], overriding.
22                 Npred[sy]:= Npred[sy]+1       // Increment Npred[sy] = |pred(sy)|.
23             END IF
24         END FOR
25         IF Npred[y] = 1 THEN                  // y is immediately dominated by idom[y].
26             push y to Stack.                  // Put y onto stack by post-DFSN order.
27         ELSE                                  // Predecessor of y is not unique.
28             Clear idom[y]                     // Idom(y) is unknown here.
29         END IF
30     END FOR
31 // (3) Loop for Reduction.              /* Reduce vertices in descending post-order DFSN. */
32     xlabl = DEFAULT                           // Clear label of x for current succx.
33     WHILE Stack is not empty DO               // Do while the stack is not empty.
34         pop y from Stack                      // Pop y to be reduced to idom(y).
35         x   := idom[y]                        // x idom y.
36         IF x not equals xlabl THEN            // Update succx for new x
37             On succx[sx] FOR ALL sx in S(x)   // Set adress of sx in S(x) into succx[sx]
38             xlabl = x                         // Set x into label of succx
39         ENDIF
40         S(x) := S(x)-y                        // Off succx[y] and delete sy in S(x)
41         FOR ALL sy in S(y) DO                 // Search all successor sy of y.
42             IF sy in S(x) THEN                // Test if succx[sy] is On.
43                 IF edge y to sy is not a back edge THEN // Select edge of DAG.
44                     Npred[sy] := Npred[sy]-1  // Decriment |pred(sy)| of sy.
45                     IF Npred[sy] = 1 THEN     // Means pred(sy)={x}.
46                         push sy to Stack      // New candidate of reduction.
47                         idom[sy] := x         // Current x must be the idom(sy).
48                     END IF
49                 END IF
50             ELSE                              // sy has not been in S(x) yet.
51                 IF x not equals idom[sy] THEN // Prevent critical appending.
52                     S(x) := S(x)+sy           // Append sy to S(x).
53                     On succx[sy]              // Set adress of sy in S(x) into succx[sy]
54                 END IF
55             END IF
56         END FOR                      // Now S(x) and succx for xlabl is updated.
57     END WHILE
58 // (4) Delete Dummy Vertex.             /* Get the tree and frontiers for original CFG. */
59     FOR ALL y in V DO
60         IF sy in S(y) is Dummy Vertex THEN
61             S(y) := S(y)-sy                   // Delete Dummy Vertex in frontier.
62         END IF
63     END FOR
64     FOR ALL v in V DO
65         IF v is Dummy THEN
66             Delete S(v)                       // Delete frontier of Dummy Vertex.
67             V := V-v                          // Delete Dummy Vertex
68     END FOR
69 END PROC
```

図 7 支配木と支配辺境の検出
Fig. 7 Identifying dominator tree and dominance frontiers.

等しい．

4.3 可約な CFG の縮約過程

図 5 に，可約な CFG の縮約過程を示す．左端の CFG では，B をヘッダとするループと，その中の C から P への飛び越しがある．このグラフの深さ優先探索順が A，B，C，D，P であるとする．ここで，B，C，D が単先行頂点なので，L18-30 で，この順にスタックに積まれる．縮約のループ L33-57 に入る

とスタックの上端から D をおろして C に縮約する．すなわち $x = C$，$y = D$ である．その結果 $S(C)$ から D が消え，さらに $S(D)$ との集合和 $\{B, P\}$ となる．このとき，$D \to P$ が $C \to P$ となり，もともとの $C \to P$ と重複するので 1 本化し，P の先行頂点数を L44 で 1 つ減らす．その結果 P の先行頂点数が 1 になり，L45-48 で判定を受けて P がスタック上端に積まれる．これは，グラフの縮約を進めるうえで基本的な機能である．逆辺を含むすべての辺は，後続頂点集合の集合和の計算に従い，縮約先の頂点に渡される．しかし先行頂点数の増減と判定に関しては DAG の辺だけを考慮する．次に G_1 では P がスタックからおろされて，縮約の対象となる．以下同様に縮約が進行するが，G_2 の縮約（$x = B$，$y = C$）で，逆辺 $C \to B$ が G_3 での自己ループ $B \to B$ に変わる．G_3 の縮約（$x = A$，$y = B$）では，B を A に縮約することで，$S(A)$ から B が消える．ここでは，$S(B)$ の中の $\{B\}$ は $A = idom(B)$ であり，L51 の判定により $S(A)$ に渡すことを止める．

4.4 非可約な CFG の縮約過程（1）

図 6 に，非可約な CFG の縮約過程を示す．左端の CFG では頂点 G と頂点 H が非可約なループを形成し，同じループの 2 つのヘッダになっている．プリヘッダは共通の頂点 F である．H の後続頂点として Q があり，グラフの入り口からの飛び越しがある．このグラフの深さ優先探索順は E，F，H，Q，G，または E，F，G，H，Q 等，複数ある．本算法ではどれを選んでも結果は同じであるので，E，F，G，H，Q とすると，単先行頂点は F，G，H の順に積むことになる．縮約のループ L33-57 では，まず H をスタックからおろして，F に縮約する．その結果はグラフ G_1 である．縮約により，$S(F) = \{G, H\}$ から H が消えて $S(F) = \{G\}$ となり，さらに $S(H) = \{G, Q\}$ から Q を受け入れて，$S(F) = \{G, Q\}$ となっている．$S(H)$ の G はスタックに G が積まれているから，$idom(G)$ は分かっている．それが F であるので L51 の判定により，$S(F)$ には $S(H)$ の G が渡されない．G_1 で $S(F) = \{G, Q\}$ となっているが，この G はもともと $S(F)$ にあった DAG 辺 $F \to G$ を示すものである．G_1 での縮約では，G が F に縮約される．これにより，$S(F) = \{G, Q\}$ から G が消える．$S(G)$ の H は縮約済みで，$idom(H) = F$ であると分かっているので，L51 の判定で H は $S(F)$ に渡されない．結局 $S(F) = \{Q\}$ となって G_2 に移る．以後は DAG の縮約になり，非可約な CFG 特有の問題はない．非可約なループのヘッダをプリヘッダに縮約する G_0 および G_1 の縮約で，ヘッダがすでにスタックに積まれていれば以上の処理が行われる．

4.5 非可約な CFG の縮約過程（2）

各ヘッダが初期のスタックには積まれていず，縮約の過程でスタックに積まれる場合は，少し違った経過になるが，結果は同じになる．上記の例ではグラフ縮約前の時点でヘッダがスタックに積まれるので，以下の説明には正確には該当しない．しかし，しいてヘッダがスタックに積まれていないと仮定し，後続頂点の操作について，以下の例に使う．まず H が縮約の過程で縮約可能になり F に縮約するとして，そのとき G はスタックに積まれていないとすると，$S(H)$ の G が $S(F)$ に渡される．$S(F)$ にはすでに G があるので，2 重化する辺 $F \to G$ を 1 本化する．これに対応して G の先行頂点数を減らすところであるが，$H \to G$ は逆辺であり，DAG 辺ではない．したがって，先行頂点数には関係させない．結局 $S(F) = \{G, H\}$ が $S(F) = \{G, Q\}$ となる．次に G を F に縮約可能になり縮約を実行すると，$S(G)$ の H は先に縮約を実行しているから，当然 $idom$ に格納済みであるので，L51 の判定により，$S(G)$ の H は $S(F)$ に渡されない．G の F への縮約により，$S(F) = \{G, Q\}$ が $S(F) = \{Q\}$ に変わる．これも前の例と同じ結果である．このように，わずかな違いはあるが結果の後続頂点集合は等しくなる．最初にヘッダがスタックに置かれない状態は，プリヘッダが複数の場合に起きる．（図 3 の (c)）

4.6 仮想頂点の挿入

図 3 にヘッダが 2 個，プリヘッダが 2 個の非可約なループの例を示す．図の (a) はもとの CFG, (b) は仮想頂点なしで DAG 化した場合，(c) は仮想頂点を追加して DAG 化した場合のグラフである．それぞれの下に，支配木を対応させて示した．(c) は支配木の葉 d を除去すれば (a) の支配木に等しくなるが，(b) はヘッダの直接支配頂点が正しくないので利用できない．これは，単なる DAG 化ではヘッダ相互間の到達可能性を失ったことを補う情報がないためである．支配木や支配辺境を求めるためには，ヘッダの直接支配頂点が正しく求められれば十分なので，Ramalingam[12) の提案による仮想頂点を用い，(c) のようにすべてのプリヘッダからすべてのヘッダへの到達を可能にする補助のグラフを追加すればよい．このときの頂点 $v(d)$ を仮想頂点と呼び．ループが非可約で，かつ，プリヘッダが複数のとき必要になる．プリヘッダが 1 つのときはそれが $v(d)$ の役割を果たすので，$v(d)$ の追加はいらない．この追加をすれば縮約は可約なグラフと同様

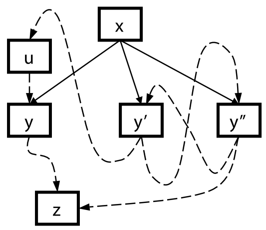

図 8 定理 2 の参考図
Fig. 8 Reference diagram for theorem 2.

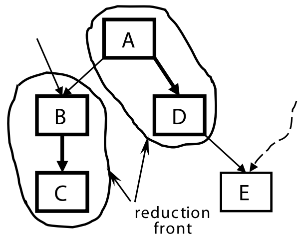

図 9 リダクションフロント
Fig. 9 Reduction front.

に行うことができ，支配木と支配辺境から仮想頂点を除去すれば，もとのグラフの支配木と支配辺境が得られる．

5. 定　　理

以下の定義および定理により本アルゴリズムの正しさを証明する．

まず，CFG の DAG 化に関連する記号を定める．

定義 1 任意の CFG の逆辺を無視して得られる DAG を，$Gd = (V, E, s)$ で表し，Gd における頂点 $v(w)$ の先行頂点を $predD(w)$，後続頂点を $succD(w)$ で表す． □

DAG において，頂点 x が頂点 y のただ 1 つの先行頂点であれば，x は y の直接支配頂点である．

定理 1 DAG $Gd = (V, E, s)$ において，$predD(y) = \{x\}$ ならば，$x\ idom\ y$ である． □

証明 1 $predD(y) = \{x\}$ ならば $x \to y \in E$．かつ，$\{u \to y \,|\, u \neq x\} = \phi$．また，$Gd$ にはサイクルがないので，$y\ dom\ x$ はありえない．よって経路 $s \Rightarrow y$ はすべて $v(y)$ に到達する直前に $v(x)$ を通る．したがって $x\ idom\ y$ である． □

DAG において，頂点 x が他の頂点を 1 つでも直接支配していれば，頂点 x をただ 1 つの先行頂点とするような頂点 y が少なくとも 1 つは存在する．

定理 2 $Gd = (V, E, s)$ において，$\{w \,|\, x\ idom\ w\} \neq \phi$ ならば，$\{y \,|\, predD(y) = \{x\}\} \neq \phi$ である． □

証明 2 $z \in \{w \,|\, x\ idom\ w\}$ とすると，$s \Rightarrow z$ はすべて $v(x)$ を通り，次に $\{v(y) \,|\, y \in succD(x)\}$ のいずれかを通って $v(z)$ に至る．よって経路 $s \Rightarrow z$ は $s \Rightarrow x \to y \Rightarrow z$ となる．ここで $Y \equiv \{y \,|\, s \Rightarrow x \to y \Rightarrow z\} \subseteq succD(x)$ とする．$y \in Y$ なら $x \in predD(y)$ であるから，ある $y \in Y$ について $predD(y) = \{x\}$ ならばただちに定理がなりたつ．逆に，すべての $y \in Y$ について $predD(y) \neq \{x\}$ ならば，$\{v(u) \,|\, u \in predD(y), u \neq x\} \neq \phi$ であり，経路 $u \to y \Rightarrow z$ がある．$s \Rightarrow u$ が $v(x)$ を通らなければ，$s \Rightarrow u \to y \Rightarrow z$ が $v(x)$ を通らずに $v(z)$ に達するので $x\ idom\ z$ に反する．よって $s \Rightarrow u$ はすべて $v(x)$ を通り，$v(x)$ の次に $\{v(y') \,|\, y' \in succD(x)\}$ のいずれかを通る．すなわち経路 $s \Rightarrow u \Rightarrow y \Rightarrow z$ は経路 $s \Rightarrow x \to y' \Rightarrow u \to y \Rightarrow z$ でなければならず，$y' \in Y$ である．したがって y'，$y \in Y$ の間に経路 $y' \Rightarrow y$ が存在する．すべての $y \in Y$ について $predD(y) \neq \{x\}$ ならば，Y の一部または全部を通るサイクルが少なくとも 1 つ存在する（**図 8**）．これは CFG Gd が DAG であることに反するので．いずれかの $y \in Y$ について $predD(y) = \{x\}$ でなければならない．よって $\{y \,|\, predD(y) = \{x\}\} \neq \phi$ である． □

DAG において，頂点 y の先行頂点が頂点 x ただ 1 つであり，y の後続頂点がないか，または後続頂点があっても，y をただ 1 つの先行頂点とする後続頂点がないとき，そのような頂点 y にリダクションポイント，有向辺 $x \to y$ にリダクションフロントの名称を与える．この条件を満たす頂点 y は，これを縮約することで，グラフの継続的な縮約を可能にする．

定義 2 DAG $Gd = (V, E, s)$ の頂点 $x, y \in V$ が，$predD(y) = \{x\}$ であるとき，$succD(y) = \phi$ であるか，または $succD(y) \neq \phi$ で，すべての $z \in succD(y)$ が，$|predD(z)| > 1$ ならば，頂点 $v(y)$ をリダクションポイント（Reduction Point）といい，$\{x \to y \,|\, y \in RDP(Gd)\}$ をリダクションフロント（Reduction Front：**図 9** 参照）と呼ぶ．リダクションポイントの集合，リダクションフロントの集合をそれぞれ $RDP(Gd)$，$RDF(Gd)$ で表す．$RDP(Gd)$ は縮約対象頂点の集合であり，$RDF(Gd)$ はそれらの頂点の縮約先頂点を明示する表現である． □

リダクションポイントは直接支配頂点が明示的に確定しており，かつ現在の DAG において他の頂点を支配していないことを正確に述べる（図 9 参照）．

定理 3 DAG $Gd = (V, E, s)$ において，$x \to y \in RDF(Gd)$ ならば，$x\ idom\ y$，かつ $\{w \,|\, y\ idom\ w\} = \phi$ である． □

証明 3 $Gd = (V, E, s)$ において $x \to y \in RDF(Gd)$

ならば，定義 2 から $predD(y) = \{x\}$，よって定理 1 から $x\ idom\ y$ である．また，定義 2 から $succD(y) = \phi$ であるか，または $z \in succD(y)$ がすべて $|predD(z)| > 1$ である．$succD(y) = \phi$ の場合は明らかに $\{w \mid y\ idom\ w\} = \phi$ である．次に，すべての $z \in succD(y)$ について $|predD(z)| > 1$ ならば，$predD(z) \neq \{y\}$ である．したがってすべての $v(w) \in V$ について，$\{w \mid predD(w) = \{y\}\} = \phi$．よって定理 2 の対偶から，$\{w \mid y\ idom\ w\} = \phi$ である．□

任意の CFG を DAG 化すると，SCC のヘッダに関して支配関係の情報が欠落することがある．DAG 化により支配関係についての情報欠落を生じる SCC を定義して複合 SCC と名づけ，その場合の情報欠落を補償する変換（図 3 (c) 参照）を dom 変換として定義する．**定義 3**　CFG $G = (V, E, s)$ の SCC が，複数のヘッダ $v(h_1), v(h_2), \ldots$ と複数のプリヘッダ $v(p_1), v(p_2), \ldots$ を持つとき，以下ではこの SCC を複合 SCC と呼ぶ．複合 SCC G_S について 1 個の仮想頂点 $v(d)$ を追加し，仮想辺 $p_1 \rightarrow d, p_2 \rightarrow d, \ldots$ および $d \rightarrow h_1, d \rightarrow h_2, \ldots$ を追加する．1 つの G_S に対するこれらの追加を G_S に対する dom 変換と呼ぶ．G のすべての複合 SCC について dom 変換を施すことを G の dom 変換と呼び，記号 Gc を用いて $Gc = dom(G)$ と表す．$Gc = dom(G)$ のとき，Gc における頂点 $v(w)$ の先行頂点を $predC(w)$，後続頂点を $succC(w)$ で表す．□

CFG の DAG 化操作を dag 変換として定義する．

定義 4　$Gc = dom(G)$ の SCC G_S の逆辺を隠蔽する操作を G_S の dag 変換と呼ぶ．Gc のすべての SCC について dag 変換を施すことを Gc の dag 変換と呼び，$Gd = dag(Gc)$ と表す．□

dom 変換により複合 SCC の支配関係が正しく保存されることを証するために，dom 変換を施す前の CFG における複合 SCC のヘッダの直接支配頂点は，どのヘッダについても等しく，その SCC に固有の 1 頂点であることを述べる．

定理 4　$G = (V, E, s)$ においてヘッダ $v(h_i)$，$1 \leq i \leq n$ とプリヘッダ $v(p_j)$，$1 \leq j \leq m$ を持つ複合 SCC $G_S = (V_S, E_S)$ の任意のヘッダ $v(h_i)$ の直接支配頂点は，$nca(v(p_j), 1 \leq j \leq m)$ である．□

証明 4　$v(h_i)$，$v(h_k) \in V_S$，$1 \leq i$，$k \leq n$ は互いに到達可能であり，任意の $v(h_i) \in V_S$，$1 \leq i \leq n$ に対して，$v(s)$ からすべての $v(p_j)$，$1 \leq j \leq m$ を通って到達する経路がある．逆に $v(s)$ から任意の $v(h_i)$ に $v(p_j)$，$1 \leq j \leq m$ のいずれをも通らずに到達する経路はない．したがって任意の $v(h_i) \in V_S$，$1 \leq i \leq n$ の直接支配頂点は $nca(p_j, 1 \leq j \leq m)$ である．□

任意の CFG である G に対して，$Gc = dom(G)$，$Gd = dag(Gc)$ とする．G の複合 SCC に追加した仮想頂点 d の Gd における直接支配頂点はその SCC の任意のヘッダ h の G における直接支配頂点に等しい．これからただちに，h の Gd における直接支配頂点も h の G における直接支配頂点に等しいことがいえる．

定理 5　CFG $G = (V, E, s)$ の複合 SCC $G_S = (V_S, E_S)$ において，複数のヘッダを $v(h_i), 1 \leq i \leq n$，複数のプリヘッダを $v(p_j), 1 \leq j \leq m$ とするとき，$Gc = dom(G)$ において G_S のために追加した仮想頂点が $v(d)$ であれば，$Gd = dag(Gc)$ における $v(d)$ の直接支配頂点は $v(h_i), 1 \leq i \leq n$ の直接支配頂点に等しく $nca(p_j, 1 \leq j \leq m)$ であり，$\{w \mid d\ idom\ w\} = \phi$ である．□

証明 5　定義 3 により，すべてのプリヘッダ $v(p_j)$，$1 \leq j \leq m$ から $v(d)$ への辺があり，それ以外に $v(d)$ を終点とする辺はない．したがって経路 $s \Rightarrow d$ はすべて $v(p_j)$，$1 \leq j \leq m$ のいずれかを通り，逆に $v(p_j)$，$1 \leq j \leq m$ のいずれも通らない経路 $s \Rightarrow d$ はない．よって，$v(d)$ の直接支配頂点は $nca(p_j, 1 \leq j \leq m)$ である．また，すべてのヘッダ $v(h_i)$，$1 \leq i \leq n$ に対して定義 3 により $v(d)$ を始点とする仮想辺があるから，すべてのヘッダ $v(h_i)$ には $v(d)$ を通る経路 $s \Rightarrow d \rightarrow h_i$ があり，経路 $s \Rightarrow d$ は $v(p_j)$，$1 \leq j \leq m$ のいずれかを通るので，経路 $s \Rightarrow d \rightarrow h_i$ も $v(p_j)$，$1 \leq j \leq m$ のいずれかを通る．逆に $v(p_j)$，$1 \leq j \leq m$ のいずれをも通らない経路 $s \Rightarrow d \rightarrow h_i$ はない．また，すべての $v(h_i)$ は $s \Rightarrow d \rightarrow h_i$ のほかにも $v(p_j)$，$1 \leq j \leq m$ のいずれかを始点とする辺 $p_j \rightarrow h_i$ を持つ．したがって，ヘッダ $v(h_i)$，$1 \leq i \leq n$ の直接支配頂点は $nca(v(p_j), 1 \leq j \leq m)$ である．また，$v(d)$ の後続頂点はすべてヘッダ $v(h_i)$，$1 \leq i \leq n$ であり，それらはすべてプリヘッダを通り $v(d)$ を通らない経路 $s \Rightarrow h_i$ を持つので，$\{w \mid d\ idom\ w\} = \phi$ である．□

CFG の支配木における部分木の節点は，部分木の根に対応する頂点の支配域の頂点に 1 対 1 で対応し，逆もまた真である．

定理 6　$G = (V, E, s)$ において頂点 $v(u)$ の支配域を $DR(u)$，支配木 $T(G)$ の u を根とする部分木を Tu とすると，$w \in Tu$ ならば $w \in DR(u)$ であり，逆もまた真である．□

証明 6　u を根とする部分木 Tu において，$x \in child(u)$ は $x \in Tu$，$u\ idom\ x$ で，$u\ dom\ x$ で

ある．また，$y \in child(x)$ は $y \in Tu$，$x\ idom\ y$，$x\ dom\ y$ であり，$u\ dom\ y$ がなりたつ．この推移的な支配関係は Tu の葉 z において，$parent(z)\ dom\ z$，$child(z) = \phi$ となるまでなりたつので，すべての $w \in Tu$ は $u\ dom\ w$ である．よって $w \in Tu$ ならば $w \in DR(u)$ である．逆に $z \in DR(u)$ ならば，$u\ dom\ z$ なので，経路 $s \Rightarrow z$ は必ず u を通る．$y\ idom\ z$ とすれば，経路 $s \Rightarrow z$ は $s \Rightarrow u \Rightarrow y \rightarrow z$ となり，$u = y$ の場合を含めて $u\ dom\ y$ である．次に $x\ idom\ y$ とすれば，経路 $s \Rightarrow z$ は同様に $s \Rightarrow u \Rightarrow x \rightarrow y \rightarrow z$ となる．経路 $x \rightarrow y \rightarrow z$ の始点は単調に u に接近し，有限回の操作で経路は $s \Rightarrow u \rightarrow w \rightarrow \ldots \rightarrow x \rightarrow y \rightarrow z$ となる，このとき u を根とする枝と節点の連鎖を $u = parent(w)$，$w = parent \ldots$，$x = parent(y)$，$y = parent(z)$ と Tu の中に作ることができる．よって $w \in DR(u)$ ならば $w \in Tu$ である． □

$Gc = dom(G)$ の縮約は，Gc の中の DAG に注目して行う．そのため，$Gc = dom(G)$ の縮約を $Gd = dag(Gc)$ の縮約と関係づけて定義する．

定義 5 縮約 $Rdc(Gc, x, y)$ とは，dom 変換 $Gc = dom(G)$ に対して，$Gd = dag(Gc)$ において，$x \rightarrow y \in RDF(Gd)$ であるとき，頂点 $v(y)$ を頂点 $v(x)$ に併合することをいう．x，y を明示する必要がないときは $Rdc(Gc, x, y)$ のかわりに $Rdc(Gc)$ と書く．縮約 $Rdc(Gc, x, y)$ の結果，$succC(x) = (succC(x) \cup succC(y) - \{y\}) - \{z \,|\, x\ idom\ z, z \in succC(y)\}$，およびすべての $z \in succC(y)$ について $predC(z) = (predC(z) - \{y\}) \cup \{x\}$ とする． □

グラフ G に入口以外の頂点が存在する限り，$Gc = dom(G)$ の縮約を継続できることを証するために，Gc の 1 頂点の縮約が支配木に与える影響について述べる．

定理 7 $Rdc(Gc, x, y)$ の支配木は Gc の支配木から $v(y)$ に対する葉 y および枝 (x, y) を除去した木に等しい． □

証明 7 定義 5 から，$Rdc(Gc, x, y)$ により $v(y) \in V$ を $v(x) \in V$ に縮約するとき，$Gd = dag(Gc)$ において $x \rightarrow y \in RDF(Gd)$ であるので，定理 3 から $x\ idom\ y$ であり，$\{z \,|\, y\ idom\ z\} = \phi$ である．したがって支配木 $T(Gd)$，$T(Gc)$ において $x = parent(y)$ かつ $child(y) = \phi$ であり，節点 y は $T(Gd)$，$T(Gc)$ の葉である．縮約 $v(x) = v(x) \Leftarrow v(y)$ では，$v(y)$ を $v(x)$ に統合し改めて $v(x)$ とする．これにより $v(y)$ は CFG との結合がなくなり $x\ idom\ y$ の関係もなくなるので，対応する支配木では葉 y と枝 (x, y) が除去される．他の頂点については変化はない．Gc において $s \Rightarrow x \Rightarrow w$ であれば，縮約後も $s \Rightarrow x \Rightarrow w$ は変わらない．Gc において $s \Rightarrow y \Rightarrow w$ であれば，縮約後 $s \Rightarrow x \Rightarrow w$ となるが，$\{z \,|\, y\ idom\ z\} = \phi$ から，y が経路 $s \Rightarrow w$ から消えても，w の直接支配頂点に変わりはない．また，Gc では $x \rightarrow y$ が存在するので，経路 $p \equiv s \Rightarrow y \Rightarrow w$ があれば，p が $v(x)$ を通らなくても，別に経路 $q \equiv s \Rightarrow x \rightarrow y \Rightarrow w$ があり，縮約後 $s \Rightarrow x \Rightarrow w$ となる．したがって縮約後 $s \Rightarrow y \Rightarrow w$ から $s \Rightarrow x \Rightarrow w$ が発生しても，w の直接支配頂点に変化はない．以上から $v(x)$，$v(y)$ 以外の頂点 $v(w)$ の直接支配頂点，および支配木における $parent(w)$ は縮約によって変わらず，$T(Gc)$ から枝 (x, y) と葉 y を除去した支配木は Gc^* の支配木に等しい． □

任意の CFG の dom 変換 $Gc = dom(G)$ について，縮約の一意な系列が存在することと，およびその系列の性質を示す（図 5，図 6 参照）．

定理 8 $Gc = dom(G)$ に対して，$G_0 \equiv Gc$，$G_1 \equiv Rdc(G_0)$，$G_2 \equiv Rdc(G_1), \ldots, G_k \equiv Rdc(G_{k-1})$，$k = |V| - 1$，かつ $G_k = (V_k, E_k, s_k)$ において $V_k = \{s\}$，$E_k = \phi$，$s_k = s$ であるような，定義 5 による縮約の有限な系列 $G_0, G_1, \ldots, G_k$ が存在し，G_k において Gc は $v(s)$ に縮退する． □

証明 8 $Gc = (V, E, s) = dom(G)$ の定義 5 による縮約 $Gc^* = (V^*, E^*, s^*) = Rdc(Gc, x, y)$ は，$Gd = (V, Ed, s) = dag(Gc)$ において $x \rightarrow y \in RDF(Gd)$ であるとき，$v(x) \equiv v(x) \Leftarrow v(y)$ とする．$Gd = (V, Ed, s)$ の入口 s からはじめて，すべての頂点 $v(y) \in V$ に深さ優先探索による後付け番号 $DFSN(y)$ を与えたとする．$E \neq \phi$ ならば，つねに $succD(s) \neq \phi$ であり，$w \in succD(s)$ は $predD(w) = \{s\}$ であるから，$|predD(y)| = 1$ となる $v(y)$ がつねに存在し，$predD(y) = \{idom(y)\}$ である．その中で最大の $DFSN(y)$ を持つ頂点を $v(y)$ とする．このとき $succD(y) = \phi$ なら $y \in RDP(Gd)$ である．一方 $succD(y) \neq \phi$ のときは，もし $z \in succD(y)$ が $predD(z) = \{y\}$ とすると，$DFSN(z) > DFSN(y)$ なので，$v(y)$ が $predD(w) = \{idom(w)\}$ となる $v(w)$ の中で $DFSN(y)$ 最大であるとの仮定に反する．よってすべての $z \in succD(y)$ について $predD(z) > 1$．したがってやはり $y \in RDP(Gd)$ である．すなわち $Gd = (V_i, Ed_i, s) = dag(Gc_i = (V_i, E_i, s))$ において $|Ed_i| > 0$ であれば $x \rightarrow y \in RDF(Gd)$ が存在し，$G_{i+1} = Rdc(G_i, x, y)$ が可能である．そのような y のうち $DFSN(y)$ が最大の y を用いることにして，$G_0 \equiv Gc$ と置き，系列 $G_1 \equiv Rdc(G_0), G_2 \equiv Rdc(G_1), \ldots,$ を作れば，系列は一意に定まり，$Gd =$

$(V_k, Ed_k, s) = dag(G_k = (V_k, E_k, s))$ において $V_k = \{s\}$，$Ed_k = \phi$ となるまで縮約を継続でき，$k = |V - \{s\}| = |V| - 1$ である．□

$Gc = dom(G)$ の縮約の系列により，グラフの入口以外のすべての頂点の直接支配頂点が求められることを証するために，Gc の縮約の系列における任意の G_i で検出した直接支配頂点 $idom(y)$ はもとの Gc でも $idom(y)$ であることを示す（図 5，図 6 参照）．

定理 9　$Gc = (V, E, s)$ に対する定理 8 の縮約の系列 $G_0, G_1, \ldots, G_k$ のうち，いずれかの $G_i (0 \le i \le k)$ において $G_{i+1} \equiv Rdc(G_i, x, y)$ ならば，$G_j (0 \le j \le i)$，のいずれにおいても $x\, idom\, y$ である．□

証明 9　定理 8 の系列 $G_0, G_1, \ldots, G_k$ では，$G_0 \equiv Gc$ とおいて，$G_1 \equiv Rdc(G_0), \ldots, G_{i+1} \equiv Rdc(G_i), \ldots$，$G_k \equiv Rdc(G_{k-1})$ が進行する．$G_{i+1} \equiv Rdc(G_i) = Rdc(G_i, x, y)$ ならば，定理 7 により $T(G_{i+1})$ は $T(G_i)$ から葉 y および枝 (x, y) を除去した木に等しく，縮約 $Rdc(G_i)$ を実行するまでは頂点 $v(x)$，$v(y)$ が存在し，$x\ idom\ y$ である．系列 $G_0, G_1, \ldots, G_k$ ではそれぞれの各縮約において 1 つの頂点が縮約を受け，対応する支配木の葉がその枝とともに支配木から除去される．G_0 には Gc 全体の支配木が対応し，$G_1, \ldots, G_i$ のそれぞれで，いずれかの葉が 1 つずつ除去され，それぞれの縮約に際しては，定理 7 により除去される葉と枝以外の節点や枝の結合状況には変化がないから，G_i において頂点 $v(x)$，$v(y)$ があり，$x\, idom\, y$ であるならば，それまでに実行される $G_j (0 \le j \le i)$，のいずれにおいても．頂点 $v(x)$，$v(y)$ があり，$x\, idom\, y$ でなければならない．□

$Gc = dom(G)$ の縮約の系列により，グラフのすべての頂点の支配辺境が（空の場合を含めて）求められることを証するために，Gc の縮約の系列における任意の G_i で検出した支配辺境 $DF(y)$ は，もとの Gc でも $DF(y)$ であることを示す（図 5，図 6 参照）．

定理 10　$Gc = (V, E, s)$ に対する定理 8 の縮約の系列 $G_0, G_1, \ldots, G_k$ のうち，いずれかの $G_i (0 \le i \le k)$ において $G_{i+1} \equiv Rdc(G_i, x, y)$ ならば，G_i における $succC(y)$ は $G_j (0 \le j \le i)$，のいずれにおいても $DF(y)$ である．□

証明 10　Gc における $v(y)$ の支配辺境を $DF(y)$，支配域を $DR(y)$ とすると，$z \in DF(y)$ に対して $y' \in DR(y)$ から z への辺 $y' \to z$，があり，また $DR(w) \subseteq DR(y)$ でない支配域 $DR(w)$ が少なくも 1 つあって，$w' \in DR(w) - (DR(w) \cap DR(y))$ から z への辺 $w' \to z$ がある．y' と w' に関するこれらの条件をここでは条件：$z \in DF(y)$ という．定理 6 により $DR(y)$，$DR(w)$ はそれぞれ y，w を根とする $T(Gc)$ の部分木 Ty，Tw に対応し，$y' \in Ty$，$w' \in Tw$ である．$Gc^* = Rdc(Gc)$ で $v(y')$，$v(w')$ のいずれもが縮約を受けないときは，条件：$z \in DF(y)$ は変わらない．$v(y')$，$v(w')$ のいずれか，たとえば $v(y')$ が縮約される場合は，$v(y") \equiv v(y") \Leftarrow v(y')$，$y" = idom(y')$，$y" = parent(y')$，$DR(y) \equiv DR(y) - \{y'\}$ となる．このとき，y' を $y"$ で置きかえることで，再び条件：$z \in DF(y)$ がなりたつことがいえる．$v(w')$ が縮約される場合は，y を w に読みかえることで，条件：$z \in DF(y)$ がなりたつ．$DR(y)$ は $v(y)$ の縮約が起きる Gi までは $v(y') \in \{v(y') \,|\, y' \in DR(y), y' \ne y\}$ の縮約により縮小しながらも存在するので，条件：$z \in DF(y)$ が存続する．$G_j (0 \le j < i)$ において $v(w)$ の縮約が起きるとすると，$v(u') \equiv v(u') \Leftarrow v(w)$，$u' = idom(w)$，$u' = parent(w)$，$u' \in Tu$ となる u を根とする部分木 Tu があって，$DR(u)$ に対応し，以後 w を u で置きかえて同様の議論により条件：$z \in DF(y)$ がなりたつ．よって $G_j (0 \le j \le i)$ のすべてにおいて $z \in DF(y)$ であり，$DF(y)$ は不変である．また $G_{i+1} \equiv Rdc(G_i, x, y)$ から，G_i において $y \in RDP(G_i)$ であり，そのとき定理 3 から $\{w \,|\, y\, idom\, w\} = \phi$ なので G における $succC(y)$ は $DF(y)$ に等しい．□

$Gc = dom(G)$ の支配辺境から G の支配辺境が求められることを証するために，Gc におけるすべての支配辺境 $DF'(y)$ から（もしあれば）仮想頂点を除去した集合は，G における支配辺境 $DF(y)$ に等しいことを述べる．

定理 11　CFG $G = (V, E, s)$ に対して，$Gc = (V', E', s) = dom(G)$ とすると，$v(y) \in V$ の G での支配辺境 $DF(y)$ は，Gc での $v(y) \in V'$ の支配辺境を $DF'(y)$ として $DF'(y) - (DF'(y) \cap rep(V' - V))$ に等しい．□

証明 11　定義 3 により $Vdm = V' - V$ は $Gc = dom(G)$ で追加した仮想頂点の集合であり，$v(y) \in V$ は仮想頂点以外の頂点である．複合 SCC がないときは，$G = Gc$ で，$Vdm = \phi$ なので，$DF(y) = DF'(y) - (DF'(y) \cap rep(Vdm) = DF'(y)$ である．複合 SCC があるとき，その 1 つ G_S について仮想頂点を $v(d)$，複数のヘッダを $v(h_1), v(h_2), \ldots, v(h_n)$ 複数のプリヘッダを $v(p_1), v(p_2), \ldots, v(p_m)$ とする．定義 3 から $1 \le n_i \le n_j \le n$ として Gc では $succC(p_i) = \{d, h_j, n_i \le j \le n_j\}$ で，$|predC(d)| > 1$，$|predC(h_j)| > 1$ であるから，$DF'(p_i) = succC(p_i)$ である．また，G では $succ(p_i) = \{h_j, n_i \le j \le n_j\}$

で，$|\,pred(h_j)\,| > 1$ であるから，$DF(p_i) = succ(p_i)$ である．したがって，$DF'(y) - DF'(y) \cap rep(Vdm) = \{d, h_j, n_i \leq j \leq n_j\} - \{d\} = \{h_j, n_i \leq j \leq n_j\} = succ(p_i) = DF(p_i)$ となる．このことはすべての複合 SCC についてなりたち，プリヘッダ以外の頂点については，G における $v(u)$ とその $succ(u)$ は，Gc における $v(u)$ とその $succC(u)$ に等しい．よって定理がなりたつ． □

支配木については，Gc の支配木において，仮想頂点に対応する節点は葉であることが明らかであり，単純にこれらを Gc の支配木から除去することにより，G の支配木が得られる．

6. 計　算　量

文中では図 7 の行番号 nn を Lnn で示す．

6.1 SCC の検出と DAG 化の操作（L10-13）

L10 の深さ優先探索は $O(M)$，L11 の SCC 探索は $O(M)$ である[17]．L12 はヘッダ検出について $O(M)$，逆辺およびプリヘッダ検出について $O(M)$ である．L13 は辺の検索でたかだか $O(M)$ である．以上の L10-13 については SCC の入れ子の最大の深さまでの繰返しとなる．入れ子の深さ L はループの数を超えず，実在のプログラムでは N，M に比して通常少ないので，適当な有限の正数 K を用いて $L < K$ とすることができる．したがって L10-13 に関する計算量は $O(M)$ である．

6.2 縮約の準備の操作（L16-30）

L16 の頂点，辺の初期化は $O(N)$，および $O(M)$ である．L18-30 は全頂点について，それぞれの後続頂点を 1 回ずつ訪問するので $O(M)$ である．

6.3 縮約の操作（L32-57）

L34-56 が L33-57 の WHILE ループで $O(N)$ の繰返しを受ける．繰返し 1 回でのそれぞれの部分の計算量は，L34-36 は $O(1)$，以下，L37 は $O(|\,S(x)\,|)$，L41-56 は $O(|\,S(y)\,|)$ であるが，その他は $O(1)$ である．$O(1)$ の部分は全体で $O(N)$ の計算量を与える．$O(|\,S(x)\,|)$ と $O(|\,S(y)\,|)$ の部分は，6.5 節で述べる．

6.4 仮想頂点の削除の操作（L58-68）

L59-63 は，すべての頂点の全後続頂点を検査する，したがって $O(M)$ である．L64-68 は全頂点を検査するので $O(N)$ である．よって L58-68 については $O(MAX(N, M))$ である．

6.5 $S(x)$ と $S(y)$ に関する計算量

すべての $v \in V$ についての後続頂点の数 $|\,S(v)\,|$ が，縮約の継続中終始不変と仮定すれば，$sy \in S(y)$ の検査 L41 は全縮約過程で考えると $O(M)$ である．

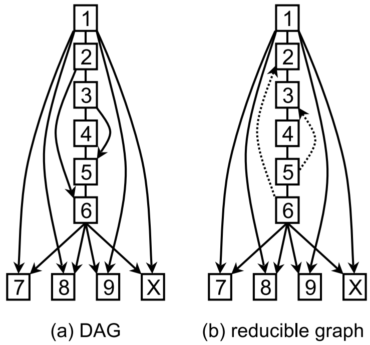

図 10 計算量増大の可能性

Fig. 10 Possibility of increasing complexity.

$sx \in S(x)$ については，sx のアドレスを $succx$ 配列に展開する操作 L37 が全縮約過程で考えれば $O(M)$ となる．本節では，この仮定を不変仮定と呼ぶ．

現実には，y の辺境 $sy \in S(y)$ が $sy \in S(x)$ でなければ，sy が $S(x)$ へ追加され $S(x)$ の増加を招く．一方，y の縮約に対応して必ず $y \in S(x)$ の削除が起きて $S(x)$ を減少させる．したがって正確な意味での不変仮定が実現する確率は低いが，辺の数 M にくらべて，y の支配辺境のうち上記の条件に合うものの数は少なく，累積されていかないので，現実の計算量は $O(M)$ に対して有限の変動が加わった値 $O(M + \alpha)$，$\alpha \ll M$ と見ることができて，結局 $O(M)$ である．

ここで，**図 10** に示すパターンを考える．図 10 (a) は，頂点 1〜6 が分岐と合流のみからなり，6 の後続頂点 7〜X は 6 にとって複数の辺境をなしている．しかし，この場合は，頂点 3，4，5 については，それぞれ 6，5，6 が頂点 1 個のみの辺境であり，頂点 7，8，9，X が計算量をふやすのは，頂点 6 を頂点 2 に縮約するときのみである．次に図 10 (b) では，頂点 7，8，9，X の 4 頂点が頂点 5 から 2 まで計 4 回コピーされ，この 4 頂点の範囲で計算量を増加させる．

以下では図 10 (b) について考える．(b) の繰返し制御では計算量の増加が起きやすい．その程度は，図のループ 2〜6 の長さと，辺境頂点 7〜X の数に比例する．ループ 2〜6 の部分の長さは，そこに含まれるループの数に依存し，辺境頂点数の上限はグラフの頂点数を N，ループ部分の頂点数を K とすれば，$N - K$ である．この場合の計算量は $O(K(N - K))$ であり，$K = N/2$ とすれば，$O(N^2)$ となって，N 一定の条件下で最大となる．実用のプログラムでは，$N = 200$ は大規模というほどでないが，その規模で，図 10 (b) のループ入れ子部分が $N/2 = 100$ 頂点，辺境頂点数 $N/2 = 100$ となる．現実のプログラムで，ループの入れ子が基本ブロック 100 個分続いたり，辺境頂点が

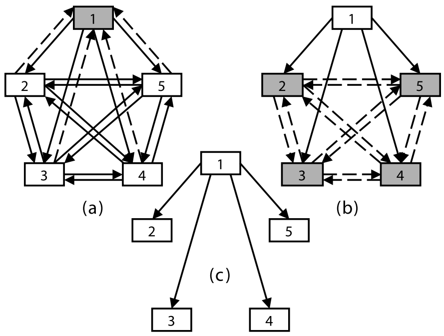

図 11 非可約な完全グラフの DAG 化
Fig. 11 DAG for irreducible complete graph.

```
DOMINATORS AND FRONTIERS:
Block| idom | Dominance Frontiers
-----+-------+---------------------
 B001| -     | B001
 B002| B001  | B003 B004 B005 B001
 B003| B001  | B004 B005 B001 B002
 B004| B001  | B001 B002 B003 B005
 B005| B001  | B001 B002 B003 B004
```

図 12 図 11 (a) の支配木と辺境
Fig. 12 Dominators and frontiers of Fig.11 (a).

100 個まとまって現れたりする可能性はほとんどない．

実際には，K または $N-K$ が，たかだか 2 桁以下の，それも低い値にとどまるであろう．その状況では計算量は $\mathrm{O}(KN)$ すなわち $\mathrm{O}(N)$ である．実在のプログラムはさまざまな構造を持っており，図 10 (b) のみで律することはできないが，このパターンは実在のプログラムにおける $\mathrm{O}(|S(x)|)$ および，$\mathrm{O}(|S(y)|)$ の計算量を考えるうえで，厳しい条件を与えるものと考えられる．

以上から，6.3 節における $\mathrm{O}(|S(x)|)$ および，$\mathrm{O}(|S(y)|)$ の計算量は，実在のプログラムでは，不変仮定の計算量からはなはだしく隔たるものでなく，全縮約過程を通じて考えて $\mathrm{O}(M)$ である．

7. 算法の実装と検証

本算法を実装し，種種の非可約なグラフに適用するとともに，SPEC CPU2000 の Fortran ソースのコンパイル出力としてのアセンブリリストから，CFG を取り出して支配関係の解析を行った．コンパイル出力を用いたのはコンパイラが非可約性を生む可能性を考慮したためである．対象とした SPEC CPU2000 の約 1,700 例の範囲では 22 件の非可約ループのヘッダが検出された．

7.1 完全グラフ

有向グラフにおける完全グラフとは，すべての頂点の間に往復の有向辺があるグラフである（**図 11**）．これを CFG と見なして，CFG の入口を頂点 1 とすれば，図 11 の (a) がオリジナル CFG であり，全体が強連結グラフであるので，入口へ向かう辺を逆辺として無視する．次に残りの CFG から，頂点 2，3，4，5 からなる入れ子の SCC が求まるので，ヘッダ 2，3，4，5 に向かう逆辺を無視すると，結局図 11 の (c) の DAG が得られる．これの縮約は簡単で，結果は**図 12** になる．頂点 1 以外はすべて頂点 1 が直接支配頂点で，また頂点 1 についてのみ自分自身が支配辺境で，他は自分以外が支配辺境である．

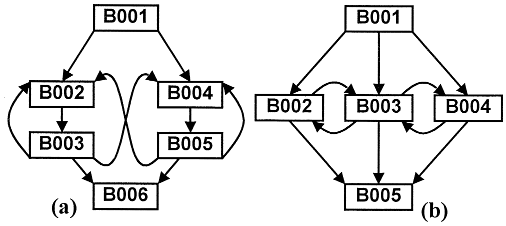

図 13 仮想の非可約な CFG の例
Fig. 13 Virtual samples of irreducible CFG.

```
DOMINATORS AND FRONTIERS:
Block| idom | Dominance Frontiers
-----+-------+---------------------
 B001| -     |  -
 B002| B001  | B006 B004 B002
 B003| B002  | B002 B004 B006
 B004| B001  | B006 B004 B002
 B005| B004  | B002 B004 B006
 B006| B001  |  -
```

図 14 図 13 (a) の支配木と辺境
Fig. 14 Dominators and frontiers of Fig. 13 (a).

7.2 仮想的非可約 CFG

支配関係を認識する機能をテストするために作った CFG のテストパターンのうち 2 例を**図 13** に示し，図 13 の (a) に対する計算結果を**図 14** に示す．

7.3 SPEC CPU2000 の例題

対象とした SPEC CPU2000 のプログラムから得た CFG は，Fortran77 から約 400 件，Fortran90 から約 1,300 件である．これらはコンパイルした結果のアセンブリリストから別のプログラムで抽出した．計算量として，SCC 検出部，縮約計算部別に，CFG の辺へのアクセス回数を記録し，別途，両者の計算部分を統合した実行時間を計時した．使用したものは 500 MHz, Pentium II のパソコンである．計時の方法は，プロトタイプからすべての外部出力機能を除去し，時間計測は，0.1 sec に達するまでの繰返し回数を求め，1 回あたりの時間を求める．入力はグラフの構造をセット

```
Language: Fortran77
Medg   µs/G    Mrf/G    µs/M Mrf/M  ITEM
 684  2500.0  2687.0  4.98  6.57  Max.
   3     7.0    10.0  1.55  3.25  Min.
  53   148.5   230.8  2.39  4.23  Mean
  66   248.2   286.8  0.56  0.47  Std.Dev.
Language: Fortran90
Medg   µs/G    Mrf/G    µs/M Mrf/M  ITEM
1608 10000.0  7415.0  6.51  7.03  Max.
   3     7.0    10.0  1.54  3.20  Min.
  65   219.9   284.9  2.43  4.21  Mean
 133   746.1   578.7  0.67  0.49  Std.Dev.
```

図 15　CPU2000 の例題による実測の統計値
Fig. 15　Statistics by samples from CPU2000.

```
Time/CFG
max.10000µsec. [µs/G]
+----------------------------------------*
|                                     ***|
|                                        |
|                                *       |
|                             *  *       |
|                                        |
|                                        |
|                                        |
|                                        |
|                                        |
|                                        |
|                 *                      |
|           * *** *                      |
|    *******                             |
|  ******                                |
*****------------------------------------+
                     max.1608 edges[Medg]
```

図 16　F90 の例題における計算時間
Fig. 16　Time for sample of program by F90.

```
Reference to Edges/CFG
max.7415 [Mrf/G]
+-------------------------------------*+
|                                      *
|                                   ** |
|                                      |
|                                      |
|                               *      |
|                            *  *      |
|                                      |
|                                      |
|                                      |
|                                      |
|              *** *                   |
|            *                         |
|         ****                         |
|   *****                              |
|*****                                 |
***------------------------------------+
                   max.1608 edges[Medg]
```

図 17　F90 の例題における辺の参照回数
Fig. 17　Access to edge in program by F90.

アップしておき，後続頂点集合のみ繰返しごとに作業域にコピーした．

図 15 は結果の統計値である．表中の記号は，Medg は CFG の辺の数，μs/G は CFG あたり実行時間，Mrf/G は CFG あたり辺アクセス回数合計，μs/M は辺あたり実行時間，Mrf/M は辺あたり辺アクセス回数を表す．

図 15 に示すように Fortran77 と Fortran90 の間には大きな差はない．平均実行時間は 2.39 μs と 2.43 μs であり，辺のアクセスは 4.23 回と 4.21 回である．

図 16，**図 17** は，Fortran90 のサンプルからの結果について，CFG ごとの実行時間と計算量（辺アクセス回数）を，CFG の辺数を横軸にとって表した散布図である．これらはいずれも等間隔座標で表示し，それぞれ [0, 最大値] を座標軸の範囲とした．単位は横軸が両図とも無次元，縦軸が図 16 については μs，図 17 については無次元である．散布図はいずれも直線の近くに分布しており，辺数 M に比例する様子が見られる．Fortran77 についてもほとんど同様の散布図が得られている．

8. おわりに

本算法によれば，非可約な CFG のプログラムでも，可約な CFG 同様，CFG の中の DAG に着目して縮約を行うことにより，簡単に支配木と支配辺境を求めることができる．非可約な CFG のうち，SCC のヘッダとプリヘッダがともに複数であるようなものについては，仮想頂点を追加し，それを介して，複数のプリヘッダと複数のヘッダを連結してから，縮約を開始する．縮約の方法は基本的に文献 19) の方法に基づいている．その方法と本算法の相違点は，支配関係の保存のために上記の処理を導入したこと，および可約，非可約に共通な問題として，逆辺を DAG の辺と区別しつつ，後続頂点への辺として平等に扱う点である．仮想頂点の追加により，すべての CFG に対してもとの CFG での支配関係を正しく保存して，支配木と支配辺境を同時に求めることが，1 つの算法で可能になった．

この算法の計算量は CFG の辺数を M として，実在のプログラムについて $O(M)$ である．算法を実装し，SPEC CPU2000 のプログラムから取得した約 1,700 件の CFG をサンプルとして，支配関係を求める計算量を測定した．測定の範囲では，計算量が $O(M)$ であることを裏づけるものであった．また CFG の有向辺 1 辺あたりの所要時間は平均約 2.4 μs であった．

謝辞　本論文に多くの貴重なご助言をいただいた，査読者，論文誌編集委員の方々に深く感謝いたします．

参考文献

1) Aho, A., Hopcroft, J. and Ullman, J.: On finding the least common ancestors in trees, *ACM Symposium on Theory of Computing*, Austin, Texas (1973).

2) Aho, A., Hopcroft, J. and Ullman, J.: *The Design and Analysis of Computer Algorithms*, Addison-Wesley (1974).

3) Aho, A.V., Sethi, R. and Ullman, J.D.: *Compilers Principles, Techniques, and Tools*, Addison-Wesley, Reading, Massachusetts (1986).

4) Aho, A. and Ullman, J.: The Theory of Parsing, *Translation, and Computing, Vol.II: Compiling*, Prentice-Hall, Englewood Cliffs. N.J. (1972).

5) Alstrup, S., Harel, D., Lauridsen, P.W. and Thorup, M.: Dominators in linear time, *SIAM J. Comput.*, Vol.28, No.6, pp.2117–2132 (Jun. 1999).

6) 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).

7) 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).

8) Havlak, P.: Nesting of Reducible and Irreducible Loops, *ACM Trans. Prog. Lang. Syst.*, Vol.19, No.4, pp.557–567 (Jul. 1997).

9) Lowry, E. and Medlock, C.: Object code optimization, *Comm. ACM*, Vol.12, No.1, pp.13–22 (1969).

10) Ochranova, R.: Finding dominators, Proc. FCT'83 Borgholm Sweden, LNCS 158 Foundation of Computation Theory, pp.328–334 (1983).

11) Ramalingam, G.: Identifying Loops in Almost Linear Time, *ACM Trans. Prog. Lang. Syst.*, Vol.21, No.2, pp.175–188 (Mar. 1999).

12) Ramalingam, G.: On loops, dominators and dominance frontiers, *Proc. SIGPLAN'00 Conf. on Programming Language Design and Implementation*, pp.233–241 (June 2000).

13) 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).

14) Sreedhar, V., Gao, G. and Lee, Y.: Identifying Loops Using DJ Graphs, *ACM Trans. Prog. Lang. Syst.*, Vol.18, No.6, pp.649–658 (Nov. 1996).

15) Steensgaard, B.: Sequentializing Program Dependence Graphs for irreducible Programs, Technical Report MSR-TR-93-14, Microsoft Research, Redmond Wash. (Oct. 1993).

16) Tarjan, R.E.: Testing Flow Graph Reducibility, *J. Comput. Syst. Sci.*, Vol.9, pp.355–365 (1974).

17) Tarjan, R.E.: Finding Dominators in Directed Graphs. *SIAM J. Comput.*, Vol.3, No.1, pp.62–89 (Mar. 1974).

18) 中田育男：コンパイラの構成と最適化，朝倉書店 (1999).

19) 齋藤鐵男，鈴木　貢，渡邊　坦：非循環グラフにおける支配関係の簡潔な検出算法，情報処理学会論文誌：プログラミング，Vol.43, No.SIG 1 (PRO13), pp.48–59 (2002).

(平成 14 年 2 月 18 日受付)
(平成 14 年 5 月 16 日採録)

齋藤　鐵男（正会員）

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），構造解析を中心に多くの応用計算システムを開発．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情報工学専攻（博士後期課程）．日本ソフトウェア科学会，電気学会各会員．

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．