Vol. 30 No. 9 情報処理学会論文誌 Sep. 1989

自動ベクトル化コンパイラのための制御関係解析法†

國枝義敏†† 津田孝夫††

自動ベクトル化技術，あるいは自動並列化技術の中では，各種の依存関係解析に関する技術が重要となる．その依存関係は，種々の変数のデータ参照関係に起因するものと，プログラムの制御の流れに起因するものとがある．本稿ではこれらの依存関係を解析する際必要となる技術のうち，制御関係解析に関連して3種類の解析方法について述べる．これらの解析は，ベクトル化可能性の判定，およびベクトル実行可能な部分のマスク生成に関する情報抽出等，複雑な制御の流れを持つ多重ループのベクトル化には不可欠である．ここに示すすべての手法は，制御の流れに関するこれらの解析を，大域的データフロー解析と同じ枠組の中で，同時に行う特徴を持つ．そのために，大域的データフロー解析の対象とするコントロールフロー・グラフでは，通常採用される基本ブロックから，特に制御の流れの分岐点ならびに合流点を切り分けてそれらを独立した頂点とし，ある仮想的に発生させた定義がどの頂点に到達するか，あるいはある頂点にどの仮想的な定義が到達するかを調査することにより，容易にしかも詳細に制御の流れを解析する．すなわち，詳細なベクトル化を目的とするコンパイラに適したアルゴリズムとなっている．これらのアルゴリズムは，独自に開発した自動ベクトル化コンパイラ中に実現され機能している．

1. はじめに

ハードウェア技術の著しい進歩に支えられ，いわゆるスーパコンピュータの性能は急速に向上してきた．その激烈な競争の中で，日本のメーカ各社は最高速度の記録を書き換えるベクトル計算機を次々に開発し，世に出してきた．そして現在，シングルプロセッサとしての最高処理速度は数 GFLOPS に達している．

これら日本のベクトル計算機は自動ベクトル化コンパイラを有している．自動ベクトル化コンパイラは，与えられた任意のプログラム内に存在するベクトル実行可能部分を抽出し，ベクトル演算器を有効に利用するベクトル命令列に翻訳する．よって，ベクトル計算機の実効的な処理速度は，この自動ベクトル化コンパイラがいかに多くのベクトル実行可能部分を抽出できるか，すなわちいかにベクトル化率を高くできるかに完全に依存している．つまり，自動ベクトル化コンパイラというソフトウェアの性能が，ベクトル計算機のハードウェアの性能を実質的に規定してしまうとみることができる．

ベクトル実行可能性，すなわちベクトル実行しても原プログラムの意味を変えないことを判断するために，自動ベクトル化コンパイラは次の2種類の依存関係を調査する．

A) **データ依存** (data dependence)

ループ中の同一変数の2出現（定義-引用，引用-定義，定義-定義）のスカラ実行の場合の実行順序が，ベクトル実行においても保存されなければならない．これは，ループの繰り返しをも考慮したデータ参照関係による依存である．

B) **制御依存** (control dependence)[1]

制御の流れにより生ずる依存，例えば if 条件節の実行は，対応する then 節および else 節の実行より必ず先行しなければならないという制約から生ずる依存である．

自動ベクトル化技術では，上記 A) のデータ依存を調べる**データ参照関係解析**と，B) 制御依存を調べる**制御関係解析**が二本の重要な柱となっている．本論文は，後者の制御関係解析のための新しい方法を提案するものであって，次の三つの事項を対象としている．

（I） 条件分岐での制御の流れによる先行関係から生ずる，中間コード群間の制御依存．

（II） ネストした if 文の各レベルの then 節および else 節に対応してマスクベクトルを生成する際，どの条件分岐により直接制御を受けるかの判定．これを**直属の条件節の解析**と呼ぶ．

（III） 前記のデータ参照関係解析において調査の対象とする同一変数の2出現の間に生ずる制御依存．

なお，上記の解析においては，for 文，while-do 文および repeat-until 文は，中間コードから抽出したコントロールフロー・グラフ（2章で詳述）では if-goto

† Methods of Analyzing Control-Flow Relations for an Automatic Vectorizing Compiler by YOSHITOSHI KUNIEDA and TAKAO TSUDA (Department of Information Science, Faculty of Engineering, Kyoto University).
†† 京都大学工学部情報工学教室

で表現されるので，これらの制御構造から生じる制御関係も，当然元来の if 文から生じる制御関係と区別なく調べられる．また上の項目(Ⅲ)は，従来から我々がC行列検査[2]と呼んでいるデータ参照関係解析を補完するものである．

実機上では，if 文は次の3種類の方式でベクトル実行される．日本のベクトル計算機はすべて，これらの3種の if 文のベクトル化方式を可能とするベクトル命令を備えたアーキテクチャとなっている[3]．

a) マスク付きベクトル命令使用：条件節の真偽値をループの各繰り返しごとに“1”，“0”のビットベクトルとして保持するいわゆるマスクベクトルを作成する．そのマスクベクトルの制御の元に，すなわちマスクベクトルが“1”あるいは“0”であるループの繰り返しの時に参照される要素のみについて，演算等を選択的に実行できる命令（マスク付き命令と呼ぶ）を使って then 節および else 節をベクトル実行する．

b) ベクトルデータ編集（収集-拡散，収集-分散，圧縮-伸長等各社で命令の呼称は異なる）命令使用：先と同じくマスクベクトルを作成し，それにより選択的に処理されるべき要素のみをベクトルレジスタ上にロードし，演算等を行う．結果も選択的にストアされる．

c) 間接参照（間接指標，リストベクトル等各社で命令の呼称は異なる）命令使用：先と同じくマスクベクトルを作成し，それにより選択的に処理されるべき要素番号のベクトルを生成し，必要な処理をこの要素番号ベクトルにより間接参照される要素列に施す．

以上3方式いずれの場合にも，まずマスクベクトルを生成しなければならない．すなわち if 文の条件節が，then 節および else 節に先行して，ループの各繰り返しについて実行されなければならないことを意味する．これが，(Ⅰ)として挙げた制御依存である．これについては3.1節で詳述する．if 文による条件分岐がループ外に飛び出すような場合，さらにこの解析は複雑となる．しかもこの解析はベクトル化可能性の判定に大きく影響する．よって，if 文のベクトル化可能性の判定には，この制御依存の解析が必須となる．また，マスクベクトルを生成するには，ある処理がネストした if 文の中で，どの if 文の then 節あるいは else 節に含まれるかを決定できなければならない．これは，(Ⅱ)として示した制御関係解析である．これについては3.2節で詳述する．以下(Ⅰ)および(Ⅱ)をまとめ**条件分岐の影響解析**と総称する．以上述べてきたとおり，この条件分岐の影響解析はデータ参照関係に起因する依存関係と共に，自動ベクトル化コンパイラに必須の技術である．

本稿では上記の3種類の制御関係解析(Ⅰ)，(Ⅱ)，(Ⅲ)を，通常の最適化コンパイラが行う**データフロー解析** (data flow analysis)[4] により同時に解析する新しい手法を提案する．この新しい手法は京都大学工学部情報工学科津田研究室において独自に開発された自動ベクトル化コンパイラ中に実現されている[2),5)]．

2. コントロールフロー解析とデータフロー解析

次章以降での説明のためにまず，本稿で提案する解析法が応用する，そして前記の自動ベクトル化コンパイラに採用されている大域的データフロー解析法[4]，ならびにコントロールフロー解析法の概念を簡単に述べる．

2.1 コントロールフロー・グラフ

今回提案する制御関係解析手法では，一般の最適化コンパイラに採用されている三つ組・四つ組レベルの中間コードを対象とした詳細な解析が可能である．これにより，より柔軟な操作が可能となり，細かな単位での部分ベクトル化が可能となる．しかし解析の処理自体は，コストを考慮し，中間コードを直接走査するのでなく，そこから制御の流れを抽出した**コントロールフロー・グラフ** (control-flow graph, 単純に flow graph とも呼ばれる)[6),7)] を対象とする．

ここでは，説明を簡単にするため以下のようなコントロールフロー・グラフを用いる（実際のコンパイラ中では，本稿とは別目的でさらに細かく分類した頂点を有する）．

(1) **頂　　点**

● **分流型頂点**　従来のデータフロー解析に用いられるコントロールフロー・グラフにおいて採用される基本ブロックから，制御の流れが複数に分かれる条件分岐のみを取り出し，1個の別の頂点とする．

● **合流型頂点**　上記の分流型頂点に対応し，複数の制御の流れが合流している部分のみを取り出し，1個の別の頂点を割り当てる．

● **単流型頂点**　従来のコントロールフロー・グラフのいわゆる基本ブロックにおいて，上述した条件分岐部分および合流部分を取り除いた完全に制御の流れが1本である頂点．具体的には，種々の演算命令系列に対

応する．手続きあるいは関数呼び出しは，実際のコンパイラ内部では特殊な1単流型頂点として独立させている．これは，データフロー解析をできるだけ詳細にする目的のためであり，本稿での解析法については単流型頂点と同様に扱えるので，以降では触れない．

●始点，終点 制御の流れが発生する頂点および消滅する頂点．与えられたプログラムから抽出されたコントロールフロー・グラフに，それぞれただ1個存在するものとする．

（2） 有 向 辺

従来のコントロールフロー・グラフと全く同じで，頂点間に存在する制御の流れの存在と方向を示す．

このように，従来の基本ブロックと異なる頂点を導入し細分類した目的は，if 文の影響解析を行う際，分流点と合流点がはっきりさせやすいからである．

図1にコントロールフロー・グラフの簡単な例を示す．図1では，頂点Dおよび頂点Fが分流型頂点であり，頂点Bおよび頂点Kが合流型頂点である．

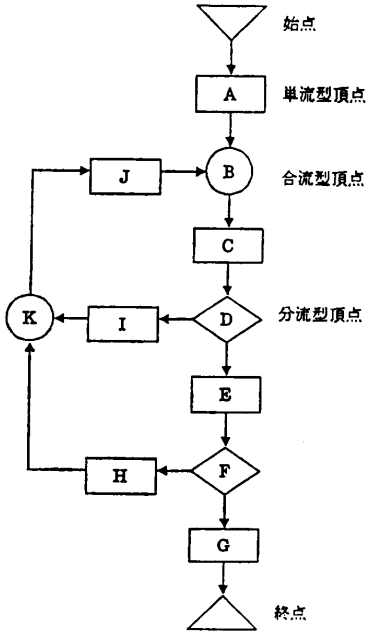


図1 コントロールフロー・グラフの例
Fig. 1 An example of a control-flow graph.

2.2 コントロールフロー解析

先の形式のコントロールフロー・グラフに対して，

（1） 各頂点間の**支配関係**，すなわち各頂点の**支配頂点** (dominator)，**直接支配頂点** (immediate dominator)[8] を求める．

（2） 各有向辺の向きを逆にした逆有向グラフを考える．分流型（合流型）頂点は合流型（分流型）頂点となる．この逆有向グラフにおいて，同じく各頂点間の支配関係，支配頂点，直接支配頂点を求める．以下，これらをそれぞれ**逆支配関係**，**逆支配頂点**，**直接逆支配頂点**と呼ぶことにする．上記のうち，直接逆支配頂点の定義は文献8) にはなく，逆支配頂点を求める際の副次的な概念となっている．これに対し，本稿で提案する手法では，直接支配頂点および直接逆支配

表1 図1を例とした支配関係および逆支配関係
Table. 1 The back dominate and forward dominate relations on Fig. 1.

頂点	支配関係		逆支配関係	
	支配頂点	直接支配頂点	逆支配頂点	直接逆支配頂点
A	始点	始点	B, C, D, E, F, G, 終点	B
B	始点, A	A	C, D, E, F, G, 終点	C
C	始点, A, B	B	D, E, F, G, 終点	D
D	始点, A, B, C	C	E, F, G, 終点	E
E	始点, A, B, C, D	D	F, G, 終点	F
F	始点, A, B, C, D, E	E	G, 終点	G
G	始点, A, B, C, D, E, F	F	終点	終点
H	始点, A, B, C, D, E, F	F	B, C, D, E, F, G, J, K, 終点	K
I	始点, A, B, C, D	D	B, C, D, E, F, G, J, K, 終点	K
J	始点, A, B, C, D, K	K	B, C, D, E, F, G, 終点	B
K	始点, A, B, C, D	D	B, C, D, E, F, G, J, 終点	J

頂点が重要な概念となる.

上記(1), (2)の解析を合わせ広く**支配関係解析**と呼ぶことにする.

支配頂点, 逆支配頂点ならびに直接支配頂点の定義[8]は以下のとおり.

[定義]

コントロールフロー・グラフ上の頂点vと頂点wにおいて, グラフの根である始点から頂点wに達するどの経路 (path) も必ず頂点vを通るとき, 頂点vは頂点wを支配する, あるいは頂点vは頂点wの支配頂点であると定義する. 同様に, コントロールフロー・グラフの逆有向グラフにおいて, 頂点vが頂点wを支配するとき, 頂点vは頂点wを逆支配する, あるいは頂点vは頂点wの逆支配頂点であると定義する. さらに, 頂点vの支配頂点の集合Dの要素である頂点d(頂点vを除く) が, 頂点v, dを除く集合Dの他のすべての要素の頂点に支配されるとき, 頂点dは頂点vの直接支配頂点という. 直観的には, 頂点vの直接支配頂点とは, 頂点vの支配頂点の中で最も頂点vに近いものである. □

表1は図1のコントロールフロー・グラフについて行った支配関係解析例である.

直接支配頂点から支配頂点 (ならびに直接逆支配頂点から逆支配頂点) を求めるため, 我々のコンパイラでは文献9) のアルゴリズムを用いている. これは本論に関係しないので説明は省略する. また, 上述の支配関係解析の後, コントロールフロー解析としては, ループ構造の検出等制御構造の解析を行うのが一般的であるが, 本稿とは直接関係しない別目的の解析であるので, これについても省略する.

2.3 大域的データフロー解析

我々のコンパイラでは文献4)の考え方に基づき以下のような**データフロー集合**と呼ばれる集合を使って, 手続き (および関数) ごとにそれら全域にわたる大域的データフロー解析を行っている. これらの集合はコントロールフロー・グラフの各頂点ごとに定義される.

1) gen [S]

ある頂点Sで定義され, その頂点Sの最後に到達する定義の集合. すなわち, 単流型頂点 S の中でも, 例えば代入文等で, ある変数が一般には複数回定義され得るが, その中で最後の定義だけが, 頂点Sの最後に到達する. これは, その最後に定義された値だけが, 頂点S以降で有効であることに着目している.

2) kill [S]

gen [S] の要素である各定義と同じ変数を定義する定義の, プログラム全域にわたる集合から, gen [S] を除いた集合. これは, 頂点Sで新しく変数が定義されるため, 他の定義が無効となることを示す.

3) in [S]

ある頂点Sの入口に到達する定義の集合. reach [S] とも呼ばれる. これは, 頂点Sが実行されるとき, 有効な定義を示す.

4) out [S]

ある頂点Sの出口に到達する定義の集合. これは頂点Sの実行後, 有効な定義を示す.

5) use [S]

ある頂点S内で引用が定義に先行する変数 (中間項も一時変数として含め統一的に処理している. 以下同じ) の集合.

6) def [S]

ある頂点S内で定義が引用に先行する変数の集合.

7) bin [S]

ある頂点Sの入口で保持していると思われる値を以後で引用する可能性がある変数の集合. live [S] とも呼ぶ.

8) bout [S]

ある頂点Sの出口で保持していると思われる値を以後で引用する可能性がある変数の集合. □

これらの集合は以下のデータフロー方程式を解くことにより求められる.

$$\mathrm{out}[v]=\mathrm{in}[v]-\mathrm{kill}[v]\cup\mathrm{gen}[v] \quad (1)$$

$$\mathrm{in}[v]=\cup\,\mathrm{out}[P] \quad (P: \text{頂点}v\text{の直前の頂点}) \quad (2)$$

$$\mathrm{bin}[v]=\mathrm{bout}[v]-\mathrm{def}[v]\cup\mathrm{use}[v] \quad (3)$$

$$\mathrm{bout}[v]=\cup\,\mathrm{bin}[P] \quad (P: \text{頂点}v\text{の直後の頂点}) \quad (4)$$

具体的には, 次のように求める.

(1) コントロールフロー・グラフを深さ優先でなぞり, 各頂点に深さ優先の順序づけ (depth first ordering) を行う.

(2) 各頂点ごとに, gen 集合, kill 集合, def 集合, use 集合を求める. 他の集合を空集合とする (実際には, 早く収束するよう, 例えば, out 集合には gen 集合をコピーしている).

(3) 上記のデータフロー方程式の式(1), 式(2)については深さ優先順で, 式(3), 式(4)についてはその逆順で, 変化がなくなるまで, 繰り返し in 集

合，out 集合，bin 集合，bout 集合を方程式のとおり計算する．これは，各集合が漸近的に方程式の解に近づくことを利用している．

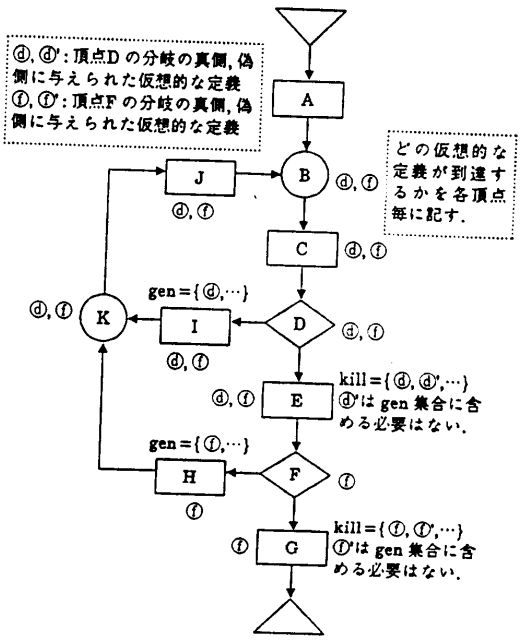
3. 制御関係解析手法

前章で述べたデータフロー集合のうちの最初の4種は，ある定義が制御の流れ（コントロールフロー・グラフの有向辺）に沿って，どの頂点に到達するか，どの頂点に到達しないかを調査するためのものである．大域的データフロー解析のこの特徴に着目し，特殊な定義を仮想的に発生あるいは消滅させることにより，第1章で述べた3種類の制御関係解析（I），（II），（III）を行うことが本論文の手法の新しい特色である．以下では，これらのアルゴリズムを説明のため個別の解析として記述するが，2.3 節で既述した大域的データフロー解析の際，すべて同時に実行できる．なお，これらのアルゴリズム記述では，すべての頂点の各データフロー集合が，この大域的データフロー解析手法にのっとり，適切に初期化されていることを仮定している．また，以下では「発生させる」とはこのデータフロー集合の gen 集合に要素として含めることをいい，同様に「消滅させる」とは kill 集合に含めることを意味するものとする．

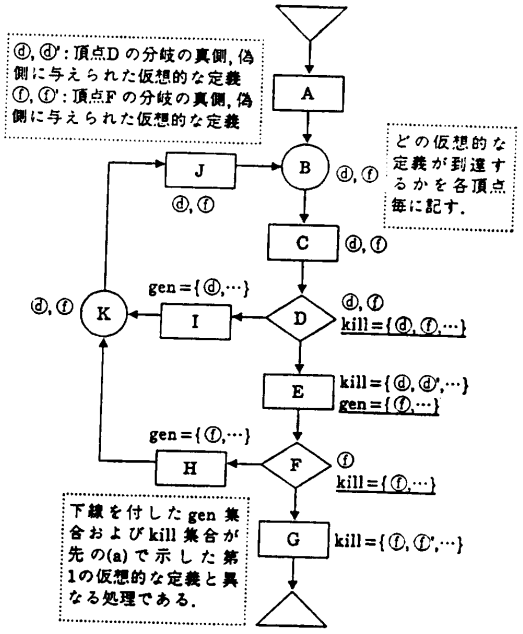
3.1 制御関係解析（I）：条件分岐から生ずる制御依存解析

まず第1章で述べたとおり，if 文のベクトル化に必要な制御の流れの分岐により生ずる依存の解析方法について述べる．この依存は，ある if 文の条件節の実行は，その制御を受ける部分すべてに対して，先行される必要があることから生ずる．よって，制御の流れを抽出したコントロールフロー・グラフにおいて，ある条件節の制御を受ける頂点をすべて選び出せば，必然的にそれらの各頂点に対応する中間コード群へ，条件節の頂点に対応する中間コードからの依存が存在すると解析できる．すなわち，コントロールフロー・グラフ上での，条件分岐の制御を受ける範囲の解析である．第1章で述べたとおり，ここでの手法は if 条件節に限らず，広く一般の条件分岐すべてを含めて統一的に解析できる．

例えば，**図2**（a）で条件分岐の一つである分流型頂点Dの制御を受ける頂点群を求めるため，仮想的な定義ⓓおよびⓓ′を頂点Dの真側および偽側それぞれに対応させ用意する．それぞれの定義を頂点Dの真側および偽側それぞれの後続の頂点Iおよび頂点Eの gen 集合に含める．すなわち，これらの頂点で発生させる．同時に，両者共に頂点Dの直接逆支配頂点Eの kill 集合に含める．すなわち，この頂点で消滅させる．次にデータフロー方程式を解く．その結果，これらの定

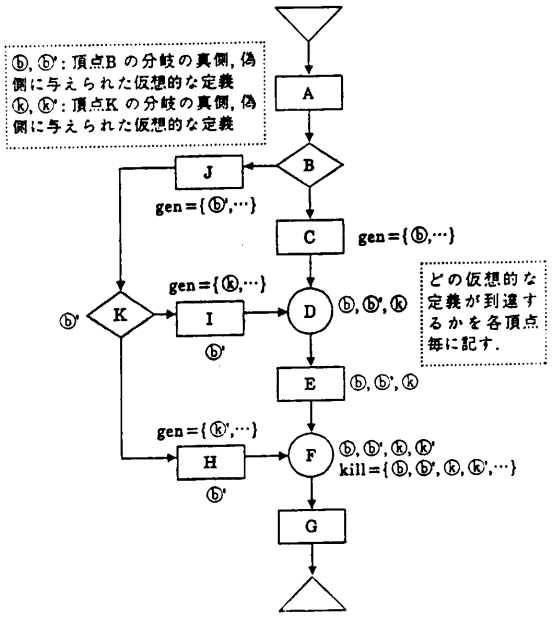


（a） 第1の仮想的な定義による制御依存解析

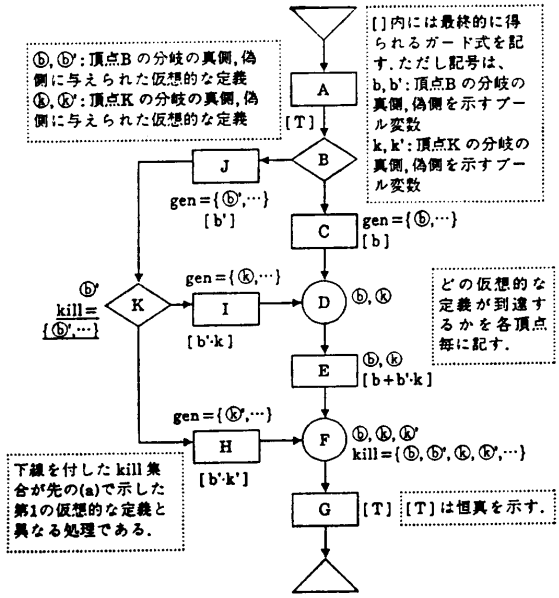


（b） 第2の仮想的な定義による直属の条件分岐解析

図2 図1の例における条件分岐の影響解析
Fig. 2 The analysis on influences of branches for Fig. 1.



(a) 第1の仮想的な定義による制御依存解析



(b) 第2の仮想的な定義による直属の条件分岐解析

図 3 if-then-else 構造でないコントロールフロー・グラフの解析

Fig. 3 The analysis of a control-flow graph with non-if-then-else structure.

義が到達する頂点（同図中ⓓと付記した頂点群），すなわち，これらの定義を in 集合の要素に持つ頂点はすべて，かつ，それらの頂点のみが頂点Dの制御を受けると判定できる．厳密な手順は以下のアルゴリズムに譲るが，図2の頂点I，頂点E等仮想的な定義を gen 集合，kill 集合に含めるべき頂点は特別に処理する．同図では同様に，頂点Fの制御依存解析のために仮想的な定義ⓕおよびⓕ′も使用している．さらに，異なるコントロールフロー・グラフでの解析例を図3(a)に示す．

[条件分岐から生ずる制御依存解析アルゴリズム]

1) 各分流型頂点にユニークな識別子を持つフラグ変数を真側出口に1個，偽側出口に1個与え，各分流型頂点からでる真側および偽側の有向辺それぞれに連なる次の頂点において仮想的に当該真側用あるいは偽側用フラグ変数に1回だけ定義が発生したものとし，同頂点の gen 集合に含める．3.2 節の制御関係解析(II)のために用いる仮想的な定義と区別するため，仮想的に発生させるこの定義を**第1の仮想的な定義**と呼ぶ．

2) 分流型頂点Pの直接逆支配頂点において，1) で用意した当該分流型頂点に対応するフラグ変数が仮想的に再定義しなおされたものとし，1) で gen 集合に含めた仮想的な定義を，この直接逆支配頂点の kill 集合に含める(消滅させる)．もちろん，この再定義自身は，以後の解析に不用であり，この直接逆支配頂点の gen 集合に含める必要はない．なお，図2(a)の頂点E，頂点Gのように定義を gen 集合に含むべき分流型頂点の次の頂点が，当該分流型頂点の直接逆支配頂点となることがある．この場合には，同図のⓓ′，ⓕ′で例示したとおり，むだな処理を避けるため，仮想的定義を gen 集合に含めない．よって，kill 集合にも含める必要はない．なお，図中でこれらの頂点の kill 集合に含めているのは，アルゴリズムのインプリメントを容易にするためである．このように，kill 集合に実際には不用な定義が含まれていても，データフロー解析の結果が誤ることはなく，また解析時間（4章参照）も増大することはない．

3) 他の定義を合わせデータフロー方程式を解く．

4) 各データフロー集合が確定した後，仮想的な定義についてのみ，各頂点の in 集合を集合 (in∪gen−kill) で置き換える．すなわち，仮想的な定義を発生させた頂点へも依存があることを統一的に処理するため gen 集合を含めて (in∪gen) とする．一方，各分流型頂点の直接逆支配頂点自身にも，仮想的な定義が到達する．ところが，ある分流型頂点の直接逆支配頂点は，その条件分岐により複数に分かれた制御の流れが再度初めて1本にまとまる頂点である．それゆえ，直接逆支配頂点自身は条件分岐の影響をもはや受けないので，依存なしとして処理する．そのため，

kill 集合の要素を取り除き，当初の in 集合にかえて (in∪gen−kill) とするのである．

5) 上記4)の置き換え後，単純に各頂点の in 集合を調査し，仮想的な定義が到達する頂点群のみが，その定義に対応する分流型頂点の制御を受けると判定できる．図2(a)の頂点D，頂点Fのようにループに含まれる分流型頂点にも，それぞれの自身の仮想的な定義が到達し，正しく判定できる． □

仮想的なフラグ変数に対する最初の定義は，当該条件分岐の制御を受ける全頂点に到達する．しかも，制御を受けない頂点には到達しない．図2(a)頂点E等の逆支配頂点自身に対する例外的な処理は上記4)の置き換えで簡単に一般の場合に含めてしまうことができる．このようにして，フラグ変数の定義が到達するかどうかを調べることにより，制御の流れによる依存関係をデータフロー解析と同時に統一的に処理し求めることが可能となる．

なお，1)では真側，偽側2種類の仮想的な定義を与えたが，単に制御依存を求めるだけであれば次の1')のように簡単化できる．

1') 各分流型頂点にユニークな識別子を持つフラグ変数1個を与え，各分流型頂点において仮想的にそれぞれの頂点に対応するフラグ変数に定義が発生したものとし，gen 集合に含める．

こうすると，仮想的定義を gen 集合に含む頂点への制御依存はないので，4) の処理では in 集合にかえて (in−kill) とするだけでよい．しかし，インプリメントにあたっては先のアルゴリズムを採用した．先のアルゴリズムは次の制御関係解析(Ⅱ)の処理と共用できる部分が多く，インプリメントが容易になるためである．また，その結果得られるより豊富な情報を基に制御関係解析(Ⅱ)の結果の検証をも行っている．

ここで提案した手法によれば，if 文によりループ外へ制御が飛び出す場合も含め，どのように複雑な制御構造を有するプログラムを与えられても正しく，しかも容易に解析できる．

3.2 制御関係解析 (Ⅱ)：直属の条件節の解析

第1章で述べたとおり，if 文のベクトル化では，ある処理がどういうマスクベクトルの制御下に実行されるのかを決定できなければならない．必要となるマスクベクトルは個々の中間コードごとに定義されるが，効率を考慮しコントロールフロー・グラフの頂点を単位に解析すれば十分である．これは，前節の制御関係解析(Ⅰ)とは逆の対応の調査である．一般に if 文がネストすることから，マスクベクトルは関係する if 条件節それぞれをブール変数としたブール式で表現できる．このブール式による表現を文献1)ではガード (guard) 式と呼んでいる．ところが，複雑に制御の流れが絡み合う任意のプログラムにおいて，ある頂点が制御を受けるこのブール式を求めることは容易ではない．3.1 節で用いた仮想的な定義からの情報を使っても，特に例えば図3のような単純な if-then-else 構造以外では，条件分岐の影響範囲のネスト構造とそれに付随するブール式を簡単には決定できない．図2のように条件分岐によりループを形成する場合には，さらに問題が複雑となる．

よって，本節では条件分岐の影響範囲のネスト構造を調査するため，コントロールフロー・グラフの各頂点において，直接制御を受ける直属の条件分岐のみをまず求め，その直属の条件分岐自身が同じく直接制御を受ける条件分岐を順にたどり，所期の目的のブール式を得る手法について述べる．この手法は，前節同様本来の目的である if による分岐に限らず，広く一般の条件分岐を対象とする．具体的には，次に示すとおり，3.1 節の仮想的な定義に似た**第2の仮想的な定義**を，各条件分岐ごとに用意して，それを用いて解析する．この第2の仮想的な定義は，3.1 節で解析に使用した定義とは全く独立である．すなわち，3.1 節で各分流型頂点ごとに真側，偽側に定義を流す2個のフラグ変数を用意したのと同様に，それらとは別にさらに2個のフラグ変数を与えるのである．これらを使い，マスク生成のための情報であるブール式を得る際に真側，偽側を区別することが可能となる．第1の仮想的な定義同様に発生および消滅させる第2の仮想的な定義が，第1の仮想的な定義と異なる点は，以下に示すとおり何度も消滅ならびに再発生を繰り返すことである．解析例図2(b)で解説する．図中の下線部が第1の仮想的な定義と異なる処理である．すなわち，仮想的な定義ⓓ，ⓕが分流型頂点Dに到達した瞬間，同頂点にてこれらの定義を kill 集合に含めて消滅させ，同時に，その分流型頂点Dの直接逆支配頂点Eにて再度 gen 集合に含める．分流型頂点Fでは到達した仮想的な定義ⓕを kill 集合に含め消滅させる．こうすることにより，多重にネストした条件節により複雑に影響を受ける場合でも，ある頂点に到達する第2の仮想的な定義は，その多重にネストした条件節の最内側の直属の条件節に対応する定義のみとなる．図3(b)では，図2とは異なるコントロールフロー・グラフにお

いて，第2の仮想的な定義がどのように処理されるかを示す．

[第2の仮想的な定義に関する処理アルゴリズム]

1) 各分流型頂点にユニークな識別子を持つフラグ変数を真側出口に1個，偽側出口に1個与え，各分流型頂点からでる真側および偽側の有向辺それぞれに連なる次の頂点において仮想的に当該真側用あるいは偽側用フラグ変数に定義が1回だけ発生したものとし，同頂点の gen 集合に含める．ただし，3.1 節の場合同様，発生させる頂点と次のステップ2) で kill 集合に含める頂点が重なる場合には，gen 集合に含めない．

2) 各分流型頂点の直接逆支配頂点において，1) で gen 集合に含めた仮想的な定義を kill 集合に含める．図2(b)で gen 集合に現れない定義を kill 集合に含めている点については，3.1 節で述べたとおりである．

3) 大域的データフロー解析を行い，仮想的な定義を流す．その際，3.1 節の第1の仮想的な定義と異なり，もし自分自身も含め，ある分流型頂点Pに到達した定義は，いったんその分流型頂点Pで kill 集合に含めて消滅させ，分流型頂点Pの直接逆支配頂点において gen 集合に含めて再度発生させる．ただし，上述の1) のステップ同様，その再発生させる頂点がステップ2) に記した本来消滅させるべき自分自身の直接逆支配頂点の場合には gen 集合に含めない．

4) 各データフロー集合が確定した後，この仮想的な定義を分流型頂点Pの kill 集合から取り除く．これは，頂点Pには，定義が到達したものとして以後の処理を統一的に行うためである．

5) その後，第1の仮想的な定義同様，仮想的な定義についてのみ，各頂点の in 集合を集合 (in∪gen−kill) と置き換える． □

ある分流型頂点Pのための第2の仮想的な定義は，上述のとおり，ある分流型頂点 P' に到達するとそこで消滅し，その分流型頂点 P' の直接逆支配頂点で再度発生する．すなわち，より深くネストした分流型頂点 P' の影響範囲内の頂点群にはこの定義は到達しない．そして，分流型頂点Pが直接影響する頂点群にのみ到達する．この性質により，各頂点ごとに先に述べた直属の条件分岐を簡単に決定できる．具体的に制御を受けるブール式を完全に求めるためには，以下の手順で行う．このアルゴリズムを述べる前に図3(b)を用い簡単に解説する．

頂点Eについて考える．頂点Eには第2の仮想的な定義ⓑおよびⓚが到達する．すなわち，頂点Eの実行を直接制御するのは，分流型頂点BおよびKであり，しかも，頂点Eはこれらの両頂点それぞれの真の側の経路に存在することがわかる．ここで，二つの分流型頂点の制御を受けるということは，両者がどちらも直属であることから，両者がネスト関係にはなく，両者から独立に制御を受けることを意味する．すなわち，(分流型頂点Bが真) OR (分流型頂点Kが真) が真の時実行されると解析できる．一方，分流型頂点Kには仮想的な定義 ⓑ′ が到達する．このことから，頂点Kは分流型頂点Bが偽の時実行されることがわかる．これらを合わせると頂点Eのガード式は図中の[ ]内に示すブール式で表されると解析できる．

[直属の条件節解析のためのアルゴリズム]

1) 第2の仮想的な定義のうち，到達する定義に対応するブール変数すべての OR をとる．もし，仮想的な定義が一つも到達しない場合には，プログラム実行において必ず1度は実行される部分であり，T (恒真) とする．この直属の条件分岐に対応するブール式を以下では単に**直属ブール式**と呼ぶことにする．

2) ステップ1) で1個以上の条件分岐の影響を受ける場合には，次のように順にそれぞれの条件分岐自身のステップ1) で求めた直属ブール式を AND で結合する．例えば，ある頂点Vの直属ブール式中に，ブール変数pが現れ，それは分流型頂点Pに対応するとすると，ブール変数pを (分流型頂点Pの直属ブール式) ・pに置き換える．ただし，この結合の際，頂点Vが分流型でループを形成し自分自身の影響を受ける場合 (図2(b)で頂点Dあるいは頂点Fがその例)，つまり $V=P$ の場合には，無限に AND 結合を続ける，すなわち無限に展開することになる．これを避けるため，このような場合には再帰的なブール表現となることを示す特別な印を付けるだけとする．

3) ステップ2) を再帰的に結合すべき変数がなくなるまで繰り返す． □

実際にコンパイラ中にインプリメントされた手法は，各頂点ごとに上記の手法を繰り返すのではなく，ステップ1) の処理のみを行っている．ステップ2) 以降の処理は部分ベクトル化を行うモジュールで，必要になった頂点のみに対して行えばよく，その場合にはループはベクトル化されるため上述のような無限展開のような状況は生じない．このループを形成する場合に，分流型頂点がそれ自身のブール変数をガード式中

に有することは，前回のループ実行時の自分自身の条件分岐の方向に，自分自身の実行がよっているという当然の事実を示すものであり，正しい解析結果である．

このように，この場合にも仮想的な定義を用いることにより，複雑に条件分岐がネストしたプログラムについても容易に解析できる．

3.3 制御関係解析（III）：同一変数の2出現間の制御の流れによる先行性解析

第1章で簡単に述べたとおり，自動ベクトル化技術の中核をなすデータ参照関係解析では，同一変数のプログラム中の2出現ごとのデータ参照関係から生じる依存関係を解析する．その際，制御の流れの解析も必要となる．例えば，あるif文のthen節とelse節に2出現が別れているような場合には，ループのある同一の繰り返しにおいて参照の衝突が生じ得ない．また，あるループ内に存在する2出現について，ループのある同一の繰り返しにおいて参照の衝突が生じた場合には，そのループ本体のみに着目して，どちらの出現が先に実行されるか，すなわちどちらの出現に制御が先に来るかによって，データ参照関係からくる依存の向きが決定される．このように，与えられた2出現に対する制御の流れによる先行性，すなわち依存関係があるのか，ないのか，また，ある場合にはどちらが先に実行されるのかという解析も欠かすことができない．詳細な部分ベクトル化を行うためには，中間コードレベルで，2出現間のデータ参照関係解析を行う必要がある．よって，2出現間の先行関係解析においても，中間コード間の先行性を解析できる必要がある．しかし，効率を考えるとコントロールフロー・グラフの頂点間で解析できれば十分である．なぜなら，コントロールフロー・グラフの同一の単流型頂点（あるいは基本ブロック）内に属する中間コード間の先行性を解析するには，単に当該頂点の先頭の中間コードから順にたどり，どちらの中間コードが先に現れるか調べるだけでよいからである．

ここでは，やはり仮想的な定義を用いることにより，データフロー集合から簡単にコントロールフロー・グラフの頂点間で先行関係を解析する手法について述べる．まず例として，図4で頂点Eと頂点Iについて考える．これらの頂点は図中に示すとおりネストレベル1のループに含まれている．しかも，そのループは頂点Dおよび頂点Kを入口に持つマルチエントリ・ループとなっている．そのために，ループがどの入口から開始されるかにより，先行関係が逆転する可能性を有する．現に，頂点Dからループが開始された場合，頂点Eが頂点Iに先行し，頂点Kから開始された場合には，逆転する．このような詳細な解析を可能とするため，ループ内の各頂点でループの入口点数と同数の仮想的な定義を独立に発生させ，これら独立な定義をそれぞれループの各入口点でkill集合に含めることにより消滅させる．図中頂点Eについては，入口点D, Kそれぞれで消滅させる仮想的な定義e_{1D}, e_{1K}を流している．これらの仮想的な定義を用いて，頂点Eにi_{1K}が到達することから，頂点Iが頂点Eに先行することを，そして頂点Iにe_{1D}が到達することから，頂点Eが頂点Iに先行することを判定できる．なぜなら，これらのループ入口点で消滅させる（kill集合に含める）仮想的な定義は，ループの繰り返しによって何度もループ内の各頂点に到達することはなく，ループの繰り返しがないとしたときの，ループ内の各頂点の先行関係にのみ依存して到達するからである．これらの定義がループを越えて流れ出すことはないため，ループ外の頂点との先行関係解析時には，ループ入口点で消滅させない仮想的な定義を別個に流す．図中．頂点Eでは定義e_0がそれに相当する．

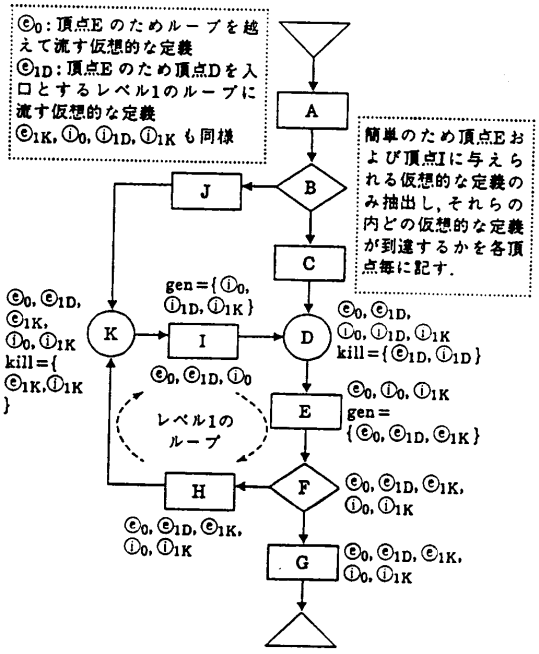


図4 複数の入口を持つループのコントロールフロー・グラフの例
Fig. 4 An example of a control-flow graph of a multi-entry loop.

[2出現間の先行関係解析アルゴリズム]

1) ある頂点Vについて，仮想的な定義を gen 集合に含める．この定義は消滅させず，終点まで流す．

2) その頂点Vがある多重ループ（特別な場合として一重の場合を含む）内に存在する時，各ループごとに当該ループの入口の数だけ仮想的な定義を発生させ，同頂点の gen 集合に含める．各ループの入口の数だけ発生させた定義は，各々当該ループの異なる入口で kill 集合に含める．

3) 大域的データフロー解析を行う．

4) 調査したい2出現が共にある同一の単流型頂点Vに存在する場合には，最悪の場合でもその頂点Vに属する中間コード列を1度走査するだけで両者の先行関係は決定できる．

5) ステップ4) 以外の一般の場合，2出現それぞれが属する単流型頂点VおよびW間の先行関係を調べる．この場合，さらに次の2とおりに場合分けする．

6) 頂点VおよびWが共通のループに属さない時，ステップ1) で発生させた両頂点の定義が相手の頂点に到達しているかどうか調べる．頂点Vで発生させた定義が頂点Wに到達すれば，頂点Vは頂点Wに先行すると判定できる．逆の場合は，逆の先行関係にあり，どちらの定義も相手に到達しなければ，両者の間に先行関係（制御依存関係）がないと判定できる．

7) 頂点VおよびWが共通のループLに属する時，ループLについてループLの入口の個数だけ発生させた両頂点の定義それぞれについて，ステップ6) と同様の判定を独立に行う．図4に例示したように複数の入口を持つループ内にあり，どの入口からループが開始されるかにより，先行関係が逆転する場合でも，正しく両方向の先行関係を調べることができる．もちろん逆転しない場合でも，正しく調査できる．これは，ステップ2) で発生させたループごとの仮想的な定義により，当該ループによる繰り返しがなかった場合，すなわち仮にループをなくした場合の制御の流れが，そのまま判定できるからである． □

この制御の流れからくる先行関係は，一般には先述の支配関係からだけでは，正しい判定が困難である．例えば，図4の頂点Cは頂点Eを支配しないが，これら2頂点間でも先行関係はある．これに対し，ここで提案した方法は，調査のコストが少なく，かつ正しく調べられるという特徴を持つ．なお，ステップ2) で用意した定義は，一部ループ外にも流れ出すが，他に影響を及ぼすことはないので，わざわざループの全出口の次の頂点において kill 集合に含め消滅させる必要はない．

4. むすび

文献1) では，制御依存をフラグ変数のデータ参照関係からくる依存ととらえ直すことにより，3.1節，3.2節と類似の解析を行っている．しかし，変換アルゴリズムが，本論文の方法に比して複雑である．また，文献1) では扱う対象がプリプロセッサであるためソーステキストレベルで変換を考えており，その方法を中間コードレベルでの詳細な解析に適用したとすると，解析のコストがさらに大きくなると予測される．これに対し，本論文で提案した解析手法は，中間コードあるいはコントロールフロー・グラフを解析対象としたコンパイラに適した方式である．さらに，通常の最適化の際に使用される大域的データフロー解析をそのまま適用するだけで，大域的データフロー解析と同時に種々の制御関係解析を行うことができる．実際にこの手法は我々が独自に開発した自動ベクトル化コンパイラ V-Pascal にインプリメントされている[2),5)]．コンパイラ V-Pascal は Pascal で記述されており，その制御関係解析部は大域的データフロー解析部と合わせ，ソーステキストで3900行強のサイズである．そのうち，仮想的な定義を発生，消滅させる部分が212行，再発生，再消滅処理が34行，in 集合の再計算が27行，3.1節で触れた検証も含めた直属の条件節解析が477行で実現されている．中核の大域的データフロー解析部は2.3節で述べたとおり，文献4)の集合演算で解析するアルゴリズムをそのままインプリメントしている．Pascal の集合演算は集合を構成する全要素のビットベクトルに対する論理演算で実現されている．よって，この手法を用いるために必要となる仮想的な定義の数（[条件分岐の数]×4+Σ{[コントロールフロー・グラフの各単流型頂点のループの深さ]×[当該ループの入口点数]+1}）は通常発生する定義の数に比較してほとんど無視できることを考え合わせると，今回提案した解析手法の実行時のコストは通常の大域的データフロー解析にほぼ完全に埋没すると考える．

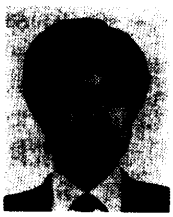
謝辞 常日頃御討論頂く本学島崎眞昭助教授，大久保英嗣助教授，岡部寿男助手はじめ津田研究室の諸氏，ならびに協同でコンパイラ V-Pascal の研究開発に努力下さった，あるいは下さるベクトル化グループの諸氏に深謝致します．本研究は一部，文部省科学

研究費補助金試験研究 (2) 63880009, 60880007 による.

参考文献

1) Allen, J.R., Kennedy, K., Portfield, C. and Warren, J.: Conversion of Control Dependence to Data Dependence, *10th Annual ACM Symposium on Principles of Programming Languages*, pp. 177-189 (1983).
2) Tsuda, T. and Kunieda, Y.: Mechanical Vectorization of Multiply Nested DO Loops by Vector Indirect Addressing, *Information Processing 86* (ed. Kugler, H.-J.), pp. 785-790, Elsevier Science Publishers, North Holland (1986).
3) 堀越 彌, 梅谷征雄: 汎用計算機のための条件制御ベクトル演算方式, 情報処理学会論文誌, Vol. 24, No. 4, pp. 531-541 (1983).
4) Hecht, M.S. and Ullman, J.D.: A Simple Algorithm for Global Data Flow Analysis Programs, *SIAM J. Comput.*, Vol. 4, pp. 519-532 (1975).
5) Tsuda, T. and Kunieda, Y.: V-Pascal: An Automatic Vectorizing Compiler for Pascal with No Language Extensions, *Proceedings of Supercomputing '88*, pp. 182-189, IEEE Computer Society (1988).
6) Allen, F.E.: Control Flow Analysis, *SIGPLAN Notices*, Vol. 5, No. 7, pp. 1-19 (1970).
7) Allen, F.E. and Cocke, J.: Graph-Theoretic Constructs for Program Control Flow Analysis, RC-3923 (#17789), pp. 1-65, IBM Thomas J. Watson Research Center, Yorktown Heights, New York (1972).
8) Tarjan, R.: Finding Dominators in Directed Graphs, *SIAM J. Comput.*, Vol. 3, pp. 62-89 (1974).
9) Lengauer, T. and Tarjan, R.E.: A Fast Algorithm for Finding Dominators in a Flowgraph, *ACM Trans. Prog. Lang. Syst.*, Vol. 1, No. 1, pp. 121-141 (1979).

(平成元年1月10日受付)
(平成元年6月13日採録)



圖枝 義敏 (正会員)

1957年生. 1980年京都大学工学部情報工学科卒業. 1982年同大学院工学研究科情報工学専攻修士課程修了. 同年4月より京都大学工学部情報工学科助手. 主として自動ベクトル化, 自動並列化に関する研究に従事. 電子情報通信学会, ACM, IEEE Computer Society 各会員.

津田 孝夫 (正会員)

1957年3月, 京都大学工学部電気工学科卒業. 現在京都大学工学部情報工学科教授, 計算機ソフトウェア講座担当. 工学博士. 自動ベクトル化/並列化コンパイラ, スーパコンピューティング, オペレーティングシステムの研究に従事. 「モンテカルロ法とシミュレーション」(培風館), 「現代オペレーティングシステムの基礎」(オーム社, 共著), 「数値処理プログラミング」(岩波書店) などの著書がある. 昭和63年度情報処理学会論文賞受賞. 本学会関西支部長. ACM, SIAM 各会員.