

到達定義・露出使用にもとづくデータ依存グラフ生成法

金田 泰* 石田 和久**

* 日立製作所 中央研究所 ** 日立製作所 ソフトウェア工場

変数参照どうしのデータ依存関係としてフロー依存、出力依存、逆依存および入力依存がある。これらをグラフ化したのがデータ依存グラフである。フロー依存は到達定義という変数定義の集合(すなわちビット・ベクトル)をつかえばもとめられることが知られているが、出力依存もそれをつかってもとめることができる。逆依存および入力依存は到達定義からもとめることができないが、この報告では露出使用という変数使用の集合をつかってもとめる方法をしめす。

これらの解析法によって、データ依存関係を高速かつすくない記憶量で計算することができる。とくに、これらの解析法の結果は Kuck らの定義による出力依存、逆依存より限定された「極小出力依存」、「極小逆依存」という関係をつかって表現することができるので、データフロー表現の記憶量をおさえることができる。

Generating Data Dependence Graphs Using Reaching Definitions and Exposed Uses

YASUSI KANADA* KAZUHISA ISHIDA**

* Central Research Laboratory, Hitachi Ltd. ** Software Works, Hitachi Ltd.

There are four kinds of variable data dependences, i.e., flow dependence, output dependence, anti-dependence and input dependence. Flow dependence can be computed by sets (bit vectors) of reaching definitions, and output dependence can also be computed by them. Though anti-dependence and input dependence cannot be computed by them, the authors have found a method for computing them with sets of *exposed uses*.

Data dependence can be computed faster by these methods with less storage. Particularly, the result of the analysis can be represented by *minimal output dependence* and/or *minimal anti-dependence*, which are restricted than general output dependence or anti-dependence, so the storage size of the dataflow representation is minimized.

1. はじめに

データ依存グラフ [1] は、スーパーコンピュータ (すなわちベクトル計算機、並列計算機など) で実行するための目的プログラムを生成する際にコンパイラがおこなうべきプログラム変換 (たとえばベクトル化) や最適化が可能かどうかを決定するなどの重要な目的でつかわれる。したがって、それを正確かつ効率よくもとめることが必要である。

データ依存グラフとは、プログラム中の各変数参照 (定義および使用) をあらわす点をつぎの 4 種類のデータ依存関係 (いずれも Kuck ら [1] による) をあらわす枝でむすんでつくった有向グラフである。

(1) フロー依存 (flow dependence)

変数定義 d から変数使用 u へのフロー依存が存在するというとき、これは d における定義値が u において使用される可能性があることをしめす。

(2) 出力依存 (output dependence)

変数定義 $d1$ から変数定義 $d2$ への出力依存が存在するというとき、これは $d1$ における定義値が $d2$ においてかきかえられる可能性があることをしめす。

(3) 逆依存 (anti-dependence)

変数使用 u から変数定義 d への逆依存が存在するというとき、これは u における使用値が d においてかきかえられる可能性があることをしめす。いいかえれば、 u へのフロー依存のうちのいずれかの始点から d への出力依存があり、 u 、 d がこの順に実行される可能性があることをしめす。

(4) 入力依存 (input dependence)

変数使用 $u1$ から変数使用 $u2$ への入力依存が存在するというとき、これは $u1$ における使用値が $u2$ においても使用される可能性があることをしめす。いいかえれば、 $u1$ へのフロー依存のうち

のいずれかの始点から $u2$ へもフロー依存があり、 $u1$ 、 $u2$ がこの順に実行される可能性があることをしめす。

これらのうち、フロー依存はいわゆる du チェイン (と ud チェインと) からなる関係である ($du = \text{definition-use}$, $ud = \text{use-definition}$)。Aho, Ullman [2] などにしめされているように、フロー依存は「到達定義」(第 2 章で定義する) を計算することによってもとめることができる。そのアルゴリズムは第 2 章でしめす。他の 3 種類のデータ依存関係のもともめかたについては、言及している文献はほとんどない。しかし、出力依存についてはフロー依存と同様の方法で到達定義からもとめることができる。これについても第 2 章でのべる。

一方、逆依存および入力依存はフロー依存および出力依存をつかってもとめることができる (その方法については第 5 章でのべる) もの、その方法では実際には存在しない依存が検出される場合があり、そのために最適化の範囲がせまられる。すなわち、満足できる逆依存および入力依存は到達定義からもとめることができない。その理由は、フロー依存および出力依存はいずれも先行する変数定義から後続する変数参照 (定義および使用) への依存であるのに対して、逆依存および入力依存は先行する変数使用から後続する変数参照への依存だからである。そこで、フロー依存や出力依存の解析における到達定義に対応する概念として「露出使用」(第 3 章で定義する) という変数使用の集合を定義することによって、フロー依存や出力依存と同様にビット列演算をつかって逆依存や入力依存をもとめることができるようにした。

この報告では、まず第 2 章でフロー依存および出力依存の解析法をしめすとともに、単純なプログラムを例としてその解析例をしめす。つぎに第 3 章で、第 2 章と対応するかたちで逆依存および入力依存の解析法をしめし、その解析例をしめす。第 4 章

では、第3章の方法と露出使用をつかわずに逆依存および入力依存をもとめる方法とを比較する。そして、第5章で結論をのべる。

2. フロー依存、出力依存の解析法

この章ではフロー依存および出力依存の解析法をしめすが、この方法はAho, Ullman [2] がのべているものとおなじである。これらをもとめるためには、あらかじめ制御フロー解析(control flow analysis) [2] をおこなう必要がある。制御フロー解析に関係する2つの用語を定義する。

定義 基本ブロック

基本ブロックとは、その途中には制御の合流がなく、また2方向以上への分岐がない範囲のことである。

定義 フロー・グラフ

フロー・グラフとは、点で基本ブロックをあらわし、点を接続する枝で基本ブロック間の制御のながれをあらわす有向グラフのことである。

制御フロー解析ではプログラムを基本ブロックに分割し、フロー・グラフをもとめる。そして、つぎのように定義される「到達定義」を制御フロー解析の結果をもちいてもとめる。

定義 到達定義 (reaching definition)

プログラムのある点における到達定義とは、プログラム中におけるすべての変数定義のうち、その定義値がその点で使用できる可能性があるもののすべてからなる集合をいう。

定義 $RIN(b)$

基本ブロック b の入口における到達定義を $RIN(b)$ とする。

定義 $ROUT(b)$

基本ブロック b の出口における到達定義を $ROUT(b)$ とする。

到達定義をもとめるために各基本ブロックについてつぎのような集合をもとめる。

定義 $RGEN(b)$

基本ブロック b にあらわれる定義のうち定義点から b の出口までのあいだでかきかえられる可能性がないものからなる集合を $RGEN(b)$ とする。

定義 $RKILL(b)$

プログラム中のいずれかの基本ブロックにあらわれる定義のうち、もし基本ブロック b に到達すれば(すなわち $RIN(b)$ にふくまれていれば) b 内でかならずかきかえられるものからなる集合を $RKILL(b)$ とする。つぎの条件は必須ではないが、解析効率をあげるために付加する： $RKILL(b)$ には、いずれかの基本ブロックの $RGEN$ にふくまれる定義だけをふくませる。

これらの集合はビット・ベクトルで表現することによって効率よく計算することができる。たとえば、図1のようなプログラムにおいては $RGEN$ および $RKILL$ は図2に示めたようになる。 $RGEN$ および $RKILL$ は基本ブロックの接続関係に依存しないので、局所的な情報だけからもとめることができる。

$RGEN$ および $RKILL$ と RIN および $ROUT$ との関係はつぎのようなデータフロー方程式によってあらわされる。

$$RIN(b) = \bigcup_{i \in \text{Pred}(b)} ROUT(i) \quad (b \text{ が入口ブロック以外するとき})$$

$$RIN(b) = \emptyset \quad (b \text{ が入口ブロックのとき})$$

$$ROUT(b) = (RIN(b) - RKILL(b)) \cup RGEN(b)$$

ここで、 $\text{Pred}(b)$ は基本ブロック b のすべての直先ブロックからなる集合を意味する。ここで直先ブロック(immediate predecessor)とは、 b への枝の始点、すなわち b の直前に実行される可能性がある基本ブロックのことをいう。たとえば、図1の基本ブ

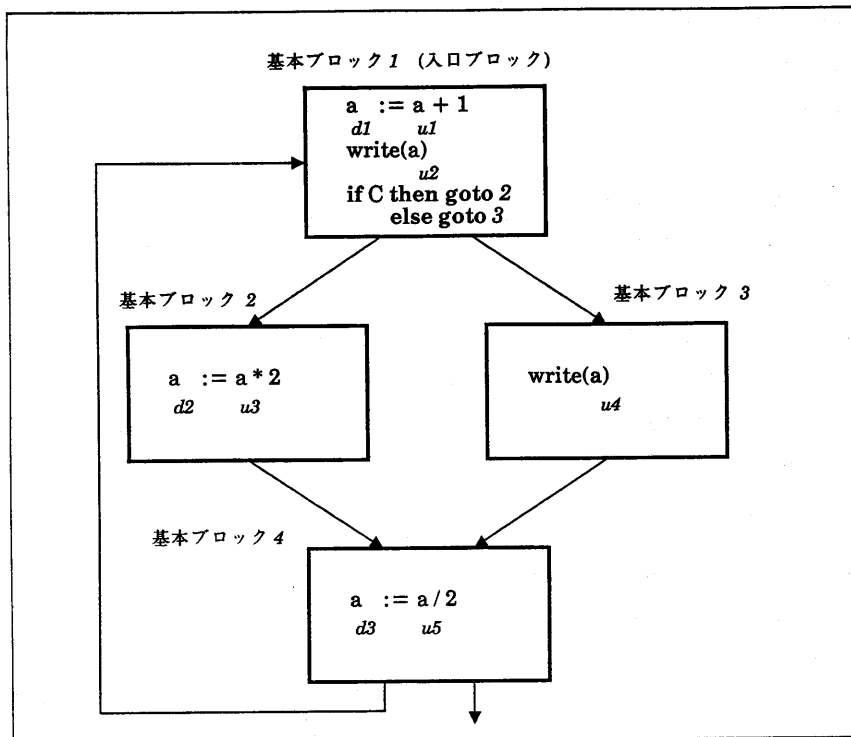


図1 プログラム例

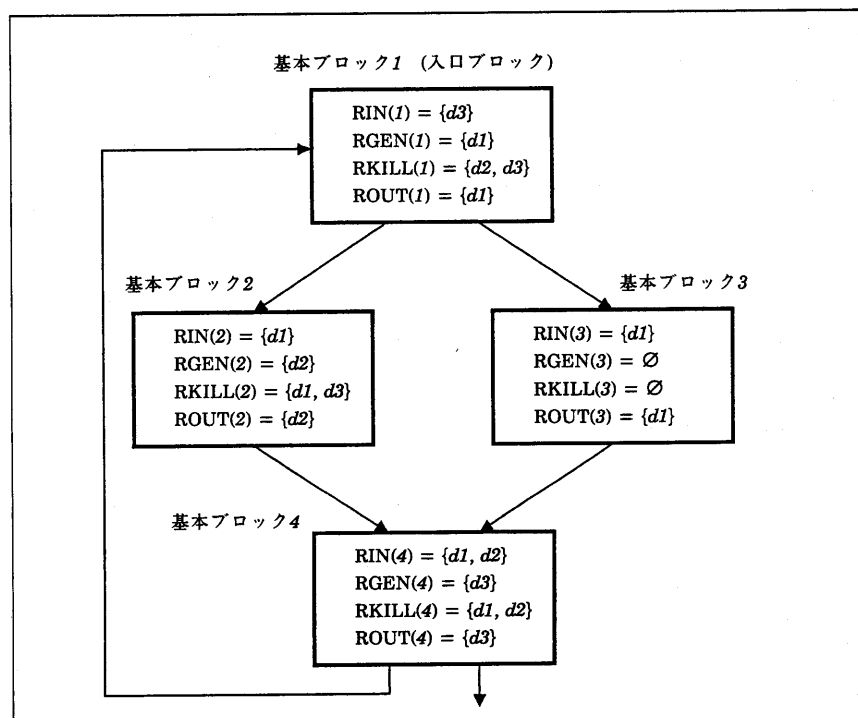


図2 図1のプログラムの到達定義

ロック4の直先ブロックは基本ブロック2および3である。

この方程式の解法としては反復解法 (iterative method) [2] と区間解析法 (interval analysis) [3] [2] とがあるが、ここでは反復解法についてのべる。反復解法においては各 RIN の初期値を空集合としてフローグラフをふかさ優先 (depth-first) にくりかえし走査しながら、図3のようにして収束するまで RIN、ROUT の値を計算する (すなわち、各到達定義の値が変化しなくなるまでくりかえす)。このくりかえし計算においては各到達定義の要素数が減少することはないので、各到達定義はかならず収束する。しかも、通常は4回以下で収束することが知られている [2]。この計算の結果、図1のプログラムの各基本ブロックにおける RIN および ROUT は図2に示めたようになる。

上記のようにして到達定義をもとめたあと、それをつかってフロー依存および出力依存をもとめるにはつぎのようにする。各変数使用について、その点での到達定義に同一変数の定義がふくまれるかどうかをしらべ、ふくまれていればその定義からその使用へのフロー依存があることにする。この計算のためには基本ブロックの入口以外の点における到達定義 (の値) が必要になるが、それはその基本ブロックの RIN をつかってもとめればよい。また各変数定義 (d とする) について、その点での到達定義に同一変数の定義がふくまれるかどうかをしらべ、ふくまれていればそれから d への出力依存があることにする。後述する理由によって、こうしてもとめられる出力依存を「極小出力依存」 (minimal output dependence) とよぶ。たとえば図1のプログラムには図4のようなフロー依存および極小出力依存が存在する。

極小出力依存と Kuck ら [1] の定義による出力依存 (以後、一般出力依存とよぶ) とのちがいはつぎのとおりである。上記の手続きでもとめられるデータ依存グラフにおいては、ある変数の定義 $d1$ 、 $d2$ 、 $d3$

が実行順でこの順にあらわれるとき、 $d1$ から $d3$ への極小出力依存は存在しない。たとえば図4においては $d1$ から $d3$ への極小出力依存は存在しない。しかし、一般出力依存はこれらのあいだに存在することになる。

すべての出力依存をもとめるかわりに極小出力依存をもとめるのは、つぎのような理由による。

- (1) 極小出力依存をたどっていくことによって、すべての出力依存をもとめることができる。なぜなら、すべての出力依存は極小出力依存の推移的閉包だからである。すなわち、定義 da と db のあいだに極小出力依存があることを $da \rightarrow_o db$ 、一般出力依存があることを $da \rightarrow_o^* db$ とあらわすとき、つぎの関係がなりたつ。

$$da \rightarrow_o^* db \Leftrightarrow$$

適当な定義 $d_1, d_2, \dots, d_n$ について

$$da \rightarrow_o d_1, d_1 \rightarrow_o d_2, \dots, d_n \rightarrow_o db.$$

(証明は省略する)。ただし、この関係がなりたつのは変数の部分的定義がないときだけである。

ここで部分的定義とは、複素数の実部への代入のような、変数の一部分の値だけをかえる操作をいう。部分的定義があるときには、上記の関係における ' $\Leftrightarrow$ ' は ' $\Rightarrow$ ' となる。

- (2) 一般出力依存をもとめるのにくらべて、極小出力依存だけをもとめるほうが、それを表現するデータが大幅に節約できる。プログラム全体にわたるデータ依存グラフがしめる記憶量はかなりの量になるから、この効果はおおきい。
- (3) 極小出力依存は、一般出力依存のなかで実行順上もっとも近接した定義どうしの関係であるから、もっとも重要である。

はじめからすべての出力依存をもとめるには、すべての基本ブロック b について $RKILL(b)$ の初期値を0として計算すればよい。ただし、このようにするとフロー依存をあわせてもとめることができなくなる。

定義 露出使用 (exposed use)

プログラムのある点における露出使用とは、プログラム中におけるすべての変数使用のうち、その使用値がその点でも使用できる可能性があるものすべてからなる集合をいう。いいかえれば、その点に達しているいずれかのフロー依存の始点(定義)からのフロー依存が達している、ともに実行される場合がある基本ブロックに属する変数使用すべてからなる集合をいう。

定義 EIN(b)

基本ブロック b の入口における露出使用を $EIN(b)$ とする。

定義 EOUT(b)

基本ブロック b の出口における露出使用を $EOUT(b)$ とする。

露出使用をもとめるために各基本ブロックについてつぎのような集合をもとめる。

定義 EUSE(b)

基本ブロック b にあらわれる使用のうち、その使用値が使用点から b の出口までのあいだでかきかえられる可能性がないものからなる集合を $EUSE(b)$ とする。

定義 EBLOCK(b)

プログラム中のいずれかの基本ブロックにあらわれる使用のうち、もし基本ブロック b に露出すれば ($EIN(b)$ にふくまれていれば) その使用値が基本ブロック b 内でかならずかきかえられるものからなる集合を $EBLOCK(b)$ とする。つぎの条件は必須ではないが、解析効率をあげるために付加する: $EBLOCK$ には、いずれかの基本ブロックの $EUSE$ にふくまれる使用だけをふくませる。

たとえば、図 1 のようなプログラムにおいては $EUSE$ および $EBLOCK$ は図 5 にしめたようになる。

$EUSE$ 、 $EBLOCK$ と EIN 、 $EOUT$ との関係はつぎのようなデータフロー方程式によってあらわされる。

$$EIN(b) = \bigcup_{i \in \text{Pred}(b)} EOUT(i) \quad (b \text{ が入口ブロック以外のとき})$$

$$EIN(b) = \emptyset \quad (b \text{ が入口ブロックのとき})$$

$$EOUT(b) = (EIN(b) - EBLOCK(b)) \cup EGEN(b)$$

この方程式は到達定義に関するデータフロー方程式とまったくおなじかたちをしているので、おなじようにしてとくことができる。図 1 のプログラムの各基本ブロックにおける EIN および $EOUT$ は図 5 にしめたようになる。

逆依存および入力依存はつぎのようにすればもとめることができる。各変数定義についてその点での露出使用に同一変数の使用がふくまれるかどうかをしらべ、ふくまれていればその使用からその定義への逆依存があることにする。また、各変数使用についてその点での露出使用に同一変数の使用がふくまれるかどうかをしらべ、ふくまれていればその使用からその定義への入力依存があることにする。出力依存と同様の理由によって、こうしてもとめられる逆依存を「極小逆依存」(minimal anti-dependence) とよぶ(入力依存に関しては「極小」ではない)。たとえば図 1 のプログラムには図 6 のような極小逆依存および入力依存が存在する。

極小逆依存と Kuck ら [1] の定義による逆依存(以後、一般逆依存とよぶ)とのちがいはつぎのとおりである。上記の手続きでもとめられるデータ依存グラフにおいては、ある変数の使用 u とその変数の定義 $d1$ 、 $d2$ が実行順でこの順にあらわれるとき、 u から $d2$ への極小逆依存は存在しない。たとえば図 6 においては $u1$ から $d3$ への極小逆依存は存在しない。

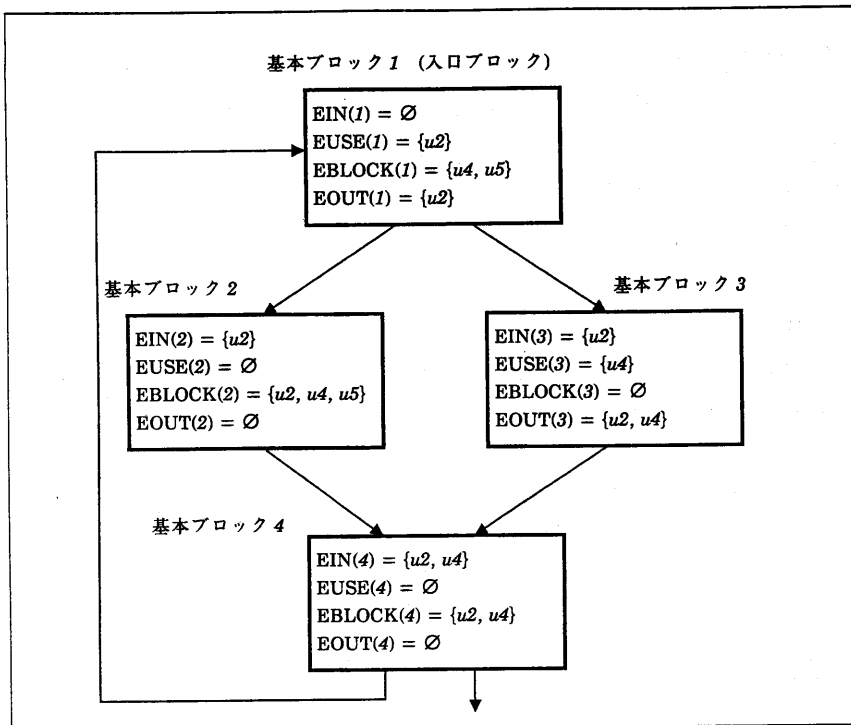


図5 図1のプログラムの露出使用

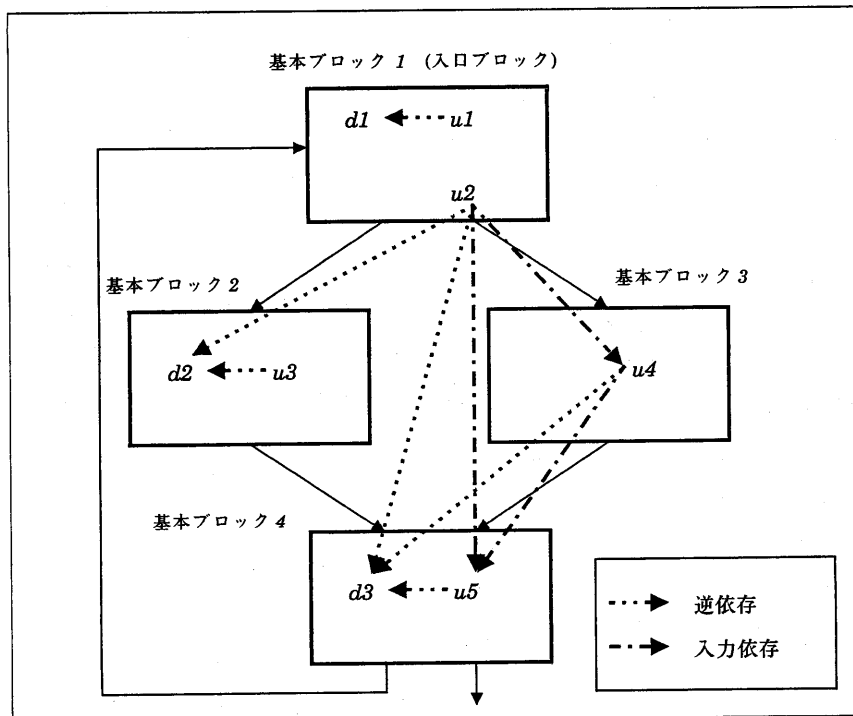


図6 図1のプログラムの逆依存および入力依存

い。しかし、一般逆依存はこれらのあいだに存在することになる。

すべての逆依存をもとめるかわりに極小逆依存をもとめる理由は、すべての出力依存をもとめるかわりに極小出力依存をもとめる理由とほぼおなじである。ただし、出力依存における理由(1)はつぎのようにかきかえられる。

(1') 極小逆依存の終点から出力依存をたどっていくことによって、一般逆依存をもとめることができる。なぜなら、使用 u と定義 d のあいだに極小逆依存があることを $u \rightarrow_a d$ 、一般逆依存があることを $u \rightarrow_{a^*} d$ とあらわすと、つぎの関係がなりたつからである。

$$u \rightarrow_{a^*} d \Leftrightarrow$$

適当な定義 $d_1, d_2, \dots, d_n$ について

$$u \rightarrow_a d_1, d_1 \rightarrow_o d_2, \dots, d_n \rightarrow_o d.$$

(証明は省略する)。ただし、この関係がなりたつのは変数の部分的定義がないときだけである。

はじめからすべての逆依存をもとめるには、すべての基本ブロック b について $\text{EBLOCK}(b)$ の初期値を $\emptyset$ として計算すればよい。また、 $\text{EBLOCK}(b)$ の定義をかえることによって「極小入力依存」をもとめることもできる。それについては付録でのべる。

4. 露出使用をつかわない方法との比較

第1章でのべたように、しかるべき仮定のもとでは露出使用をつかわずにフロー依存および出力依存から近似的な逆依存および入力依存をもとめることができる。この章ではその方法をのべるとともに、それと露出使用をつかう方法とを比較する。

まず、逆依存のもとめかたについて図7をつかってのべる。変数使用 u と変数定義 d とのあいだに逆依存があるとする。 u へのフロー依存が存在するなら、その始点である定義 ($d1$ とする) と d とのあいだには出力依存が存在する。したがって、 d への出

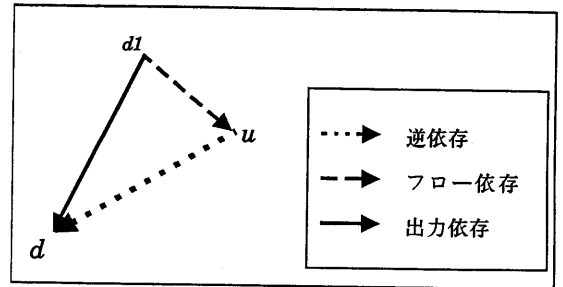


図7 露出使用をつかわない逆依存解析法

力依存を逆にたどり、その始点 (u) からのフロー依存をたどれば、その終点 ($d1$) と d との逆依存を検出することができる。

つぎに、入力依存のもとめかたについて図8をつ

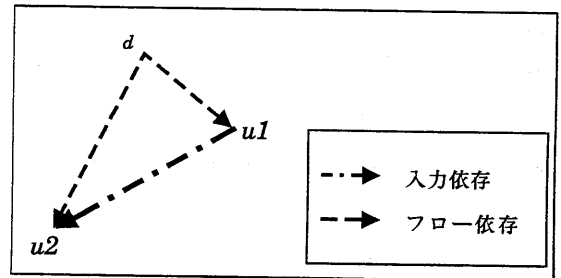


図8 露出使用をつかわない入力依存解析法

かってのべる。変数使用 $u1$ と $u2$ とのあいだに入力依存があるとする。 $u1$ へのフロー依存が存在するなら、その始点である定義 (d とする) と $u2$ とのあいだにも入力依存がある。したがって、 d へのフロー依存を逆にたどり、その始点 (d) からのフロー依存をたどれば、その終点 ($u2$) と $u1$ との逆依存を検出することができる。

しかし、上記の方法は露出使用をつかう方法にくらべてつぎのような欠点がある。

(1) 逆依存の場合も入力依存の場合も、使用 u へのフロー依存があることを仮定している。ところが u が手続きの引数であるような場合には、手続き内部だけをみるとフロー依存が存在しないことがある。このような場合には逆依存をもとめることができない。

(2) 依存の始点と終点とがあわせて実行されること
がない基本ブロックにあるときでも依存ありと
判定してまう。たとえば、図6の u_3 と u_4 とは
露出使用をつかう解析法ではただしく入力依存な
しと判定されるが、上記の方法では依存ありと判
定される。

しかし、用途によってはこのような場合に依存あり
と判定したほうがよいことがあるから、つねに欠点
であるとはいえない。たとえばある種のベクトル
計算機においては、図6の u_3 と u_4 とが同一ベクトル
レジスタを使用するようにコンパイルされると、
両者のあいだにレジスタ衝突 (register conflict)
が生じて、本来は並列に実行できるのにそれができ
なくなる。これをあらかじめ検出して同一ベクトル
レジスタを使用しないようにするためには、上
記のようにこれらのあいだに逆依存ありと判定した
ほうがよい。

5. 結論

到達定義とともに「露出使用」という変数使用の
集合をつかうことによって、(極小)出力依存と(極
小)逆依存を高速に、かつ記憶を節約しながら計算す
ることができる。極小出力依存および極小逆依存を
効率よくつかうプログラム変換や最適化は今後の課
題である。

参考文献

- [1] Kuck, D., J., Kuhn, R., H., Padua, D., H.,
Leasure, B., and Wolfe, M., Dependence Graphs
and Compiler Optimizations, 8th Annual ACM
Symposium on Principles of Programming
Languages, pp. 207-218, Jan. 1981.
- [2] Aho, A. V., Sethi, R., and Ullman, J. D.,
Compilers - Principles, Techniques, and Tools,
Addison Wesley, 1986.
- [3] Kennedy, K., A Comparison of two algorithms for
global data flow analysis, SIAM J. of Computing,
Vol.5, No.1, 1976.

付録 入力依存の記憶量を節約するための方法

第3章でのべた露出使用にもとづく解析法は、入
力依存については記憶量をへらすくふうをしていな
い。露出使用の定義をつぎのようにかえれば、それ
が可能になるとともに、逆依存の記憶量もさらにへ
らすことができる。

定義 露出使用

プログラムのある点における露出使用とは、
プログラム中におけるすべての変数使用のうち、
その使用値がその点でも使用できる可能性が
あり、定義点からみて最初の使用となる可能性が
あるものすべてからなる集合をいう。

EIN, EOUTの定義は第3章とおなじである。

定義 EUSE(b)

基本ブロックにあらわれる使用のうち、その
使用値が使用点から b の出口までのあいだでかき
かえられる可能性がなく、未使用である可能性が
あるものからなる集合を EUSE(b) とする。

定義 EBLOCK(b)

プログラム中のいずれかの基本ブロックにあ
らわれる使用のうち、その使用値が基本ブロッ
ク b 内でかならず参照される (かきかえられるか
または使用される) ものからなる集合を
EBLOCK(b) とする。

この定義にもとづいてもとめた入力依存を「極小
入力依存」とよぶ。Kuckら [1]の定義による入力依
存は極小入力依存の推移的閉包である。こうしても
とめると逆依存もさらに限定される。すなわち、定
義 d と使用 u のあいだに極小入力依存があることを
 $d \rightarrow_i u$ 、一般逆依存があることを $u \rightarrow_a^* d$ とあら
わすと、つぎのような関係がなりたつ。

$$u \rightarrow_a^* d \Rightarrow$$

適当な使用 $u_1, u_2, \dots, u_n$, 適当な定義 $d_1, d_2, \dots, d_n$ について $u \rightarrow_i u_1, u_1 \rightarrow_i u_2, \dots, u_n \rightarrow_a d_1, d_1 \rightarrow_o d_2, \dots, d_n \rightarrow_o d$ 。

(証明は省略する)。

このように入力依存を限定することができるにもかかわらず本文ではその方法をとらなかった理由はずぎのとおりである。ベクトル化や高水準の中間語(目的プログラムより原始プログラムにちかいレベルの中間語)上でおこなうべき最適化には入力依存の情報は必要ないとかんがえられる。(第4章で例としてあげたベクトルレジスタわりあても低水準の中間語上でおこなうべき操作である)。付録における方法では入力依存もあわせてもとめなければ逆依存の情報も不完全になってしまうが、第3章の方法では逆依存だけをもとめることができる。したがって、逆依存だけが必要な場合にはこのほうが記憶量がすくなくてすむ。