社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.
SS95-37 (1996-01)

再帰やポインタを含むプログラムの効率的な依存関係解析法の提案

佐藤 慎一† 植田 良一‡ 井上 克郎†

†大阪大学 基礎工学部 情報工学科
‡(株) 日立製作所 システム開発研究所
〒560 大阪府豊中市待兼山町 1-3
大阪大学 基礎工学部 情報工学科 知能情報処理学講座
Tel:06-850-6571(内線 6571)
E-mail:s-sato@ics.es.osaka-u.ac.jp

あらまし プログラムの依存関係解析は，デバッグや保守，コンパイル時の最適化などに利用されており，非常に有用である．しかし，エイリアスの影響が手続きを越えて伝わるようなプログラムの依存関係解析は容易ではない．本研究では，各手続きごとに，その内部で定義される変数などの情報をあらかじめ計算しておき，手続き呼び出しの際にはその情報を参照することにより，手続きの再解析を行なわないようにするという手法をエイリアスを含むプログラムに適用することによって，効率のよい依存関係解析を行なうことができるアルゴリズムを提案する．さらに，このアルゴリズムによって得られた解析結果からPDGを生成し，その上でスライシングを行なう手法についても述べる．

キーワード 依存関係解析，ポインタ，エイリアス，プログラム依存グラフ (PDG)，スライス

Efficient Dependence Analysis Algorithm for Programs with Pointers and Recursive Calls

Shinichi Sato† Ryouichi Ueda‡ Katsuro Inoue†

†Department of Information and Computer Sciences,
Faculty of Engineering Science,Osaka University
‡System Development Laboratory, Hitachi
1-3 Machikaneyama, Toyonaka, Osaka 560, Japan
Tel:06-850-6571(6571)
E-mail:s-sato@ics.es.osaka-u.ac.jp

Abstract

Program Dependence Analysis is useful in software debugging, maintainance and compiler optimization. However it is not easy to analyze the dependence of programs including pointers and recursive calls. In this paper, we propose an efficient program dependence analysis algorithm which would eliminate the repeated analysis efforts for the procedures called more than once. This is performed by storing the analysis results for the previous calls and then refering them. Furthermore, we present how to construct the PDG by using the information obtained with this algorithm, and how to caluculate the Program Slices on it.

key words program dependence analysis, pointer, alias, program dependence graph(PDG), program slice

1 まえがき

プログラムの依存関係解析は，プログラムのデバッグや保守，コンパイル時の最適化などに利用されており，非常に有用である．

[6] では，再帰を含むプログラムの依存関係解析を行なって PDG を作成し，そのグラフ上でスライス [5] を計算するアルゴリズムを提案している．このアルゴリズムでは，各手続きについて起こり得る副作用をあらかじめ求めておき，手続き呼び出しがあったときには，その手続きを解析し直すのではなく，あらかじめ求めておいた情報を用いて解析を行なうことにより，再帰を含むプログラムを効率的に解析している．しかし，このアルゴリズムは変数のデータ型としてスカラー型しか許しておらず，ポインタ変数などによるエイリアスを含むプログラムには対応していなかった．

本研究では，[6] で用いられていた方法を，ポインタ変数およびそれに伴うエイリアスを含むようなプログラムに適用することを考える．こうすることにより，一つの手続きが複数回呼ばれるような場合には，手続きを再解析せずにすむ場合があるため，効率の良い依存関係解析が行なえると考えられる．また，[6] と同様に，再帰を含むようなプログラムに対しても，その手続きを繰り返し解析して手続きのもつ副作用の情報を更新していくことにより，正確な解析が行なえる．

さらに，それ以外に，プログラム中の各文の解析についても独自の改良を行なった．これにより，[4] のアルゴリズムにおいて存在していたような，不正確さを避けることができる．

本稿ではこれ以降，2 節で提案するアルゴリズムの概要について述べる．そして，3 節で，本アルゴリズムによって PDG を生成する方法について，さらに，4 節ではその PDG 上でスライシングを行なう手法について述べ，5 節で適用例について述べる．

2 アルゴリズムの概要

本章では，本研究で提案するアルゴリズムの詳細について述べる．

2.1 用語

依存関係解析のために，いくつかの集合を定義する．

まず，「変数 a と b がエイリアスされている」ということを，(a,b) という二つ組で表す．エイリアスは，プログラム中のある地点で，複数の左辺値式 (変数名) が同じメモリ領域を参照している時に発生する．一般に，(a,b) が成り立つ時には，その要素を入れ換えたもの (b,a) も成り立つ．

次に，ある文 S の到達定義 (Reaching Definition，略して RD) の集合 $RD(S)$ を，以下の四つ組を要素にもつ集合と定義する．

$$\langle T, \mathtt{v}, (\mathtt{w_1}, \mathtt{u_1}), (\mathtt{w_2}, \mathtt{u_2}) \rangle$$

これは，「現在解析している手続きの入口にエイリアス $(\mathtt{w_2}, \mathtt{u_2})$ が存在し，かつ，文 S にエイリアス $(\mathtt{w_1}, \mathtt{u_1})$ が到達している時，文 T における変数 v の定義が S に到達している」ことを示す．この組の三項目 $(\mathtt{w_1}, \mathtt{u_1})$ が ϕ のときには，文 S に到達しているエイリアスにかかわらずこの RD は文 S に到達することを示す．また，この組の四項目 $(\mathtt{w_2}, \mathtt{u_2})$ が ϕ のときには，現在解析している手続きの入口に到達しているエイリアスにかかわらずこの RD は文 S に到達することを示す．

RD に加えて，ポインタ変数によるエイリアス解析のために，ある文 S の到達エイリアス (Reaching Alias，略して RA) の集合 $RA(S)$ を，以下の三つ組を要素にもつ集合と定義する．

$$\langle T, (\mathtt{w_1}, \mathtt{u_1}), (\mathtt{w_2}, \mathtt{u_2}) \rangle$$

これは，「現在解析している手続きの入口にエイリアス $(\mathtt{w_2}, \mathtt{u_2})$ が到達しているときに，文 T で生成されたエイリアスの組 $(\mathtt{w_1}, \mathtt{u_1})$ が文 S に到達している」ことを示す．この組の三項目 $(\mathtt{w_2}, \mathtt{u_2})$ が ϕ の時には，現在解析している手続きの入口において存在するエイリアスにかかわらず，エイリアス $(\mathtt{w_1}, \mathtt{u_1})$ が S に到達することを示す．

また，手続きの中で定義される大域変数や引数の変更情報を保持するために，各手続き P に関して以下のような集合を定義する．これらの集合は，[6] において用いられている集合を拡張したものである．

$SDglob(P)$
≡ { 手続き P内で必ず定義される
大域変数の集合 }

$PDglob(P)$
≡ { 手続き P内で定義される可能性のある
大域変数の集合 }

$ImUSE(P)$
≡ { 手続き P内で参照され，かつ，
その定義が Pの外部から伝わるような
大域変数の集合 }

$SDpara(P)$
≡ { 手続き Pを実行した時に必ず
更新される引数の集合 }

$PDpara(P)$

≡ { 手続き Pを実行した時に更新される可能性のある引数の集合 }

これ以外に，手続きを呼び出した時のエイリアス変更情報を保持するために，各手続き P に関して以下のような集合を定義する．

$MustALIAS(P)$

≡ { 手続き Pを実行した時に必ず生成される RA の集合 }

$MayALIAS(P)$

≡ { 手続き Pを実行した時に生成される可能性のある RA の集合 }

$preALIAS(P)$

≡ { 手続き Pの呼出側に存在するエイリアスの集合．これをもとに手続きを解析し直すかどうかを決定する．}

また，手続きの引数の中にポインタ型のものが含まれている場合，その仮の実体を表す変数を用意し，解析の際，実引数が指す領域とその変数とのエイリアスを生成する．これは，手続き呼び出しの副作用を正確に解析するためのものである [2].

さらに，手続き呼び出しがあった場合に，呼び出し時点での文番号，RD，RA を保持するためのスタックを用意する．

2.2 解析手順

プログラムの解析手順は以下のようになる．詳細は後の節でそれぞれ述べる．

1. プログラムを構文解析し，かつ，PDG のノードに切り分ける．
   - 手続きの宣言があるときには，その手続きに関して，2.1 節で述べた集合を蓄える領域を確保する．
   - 切りわけは，代入文，入力文，出力文，条件文や繰り返し文の条件判定部分，手続き呼出文などを単位としておこなう．
   - 各ノードについて，参照変数，定義される変数を求めておく．この段階では，手続き呼び出しの副作用について考える必要はない．
2. 各手続きの初期化を行なう (2.3 節参照).
3. 主手続きから解析をはじめ，各文の RD，RA を更新していく (2.4 節参照).

2.3 初期定義

まず，各手続きについて，2.1 節で述べた各集合の値を全てϕ(空集合) に初期化する．

2.4 各文の解析

解析は，主プログラムの最初の文から行う．構文解析によって切り分けられた PDG の各ノードを一つの単位として解析を行なう．各ノード S を解析するたびに，以下の集合が更新されるものとする．

RD(S) S での到達定義集合.

RA(S) S での到達エイリアス集合.

SuDEF(S) S で必ず定義される変数の集合．その要素は変数名および手続きの入口でのエイリアスとの組からなる．

PoDEF(S) S で定義される可能性のある変数の集合．その要素は変数名および手続きの入口でのエイリアスとの組からなる．

MayALIAS(S) S で生成される可能性のあるエイリアス集合．その要素は，生成されるエイリアスおよび手続きの入口でのエイリアスとの組からなる．

MustALIAS(S) S で必ず生成されるエイリアス集合．その要素は，生成されるエイリアスおよび手続きの入口でのエイリアスとの組からなる．

以下，プログラム中の各文 S ごとに，解析方法を述べる．

2.4.1 スカラー変数への代入を行なう文および式

S:a ← ···の形の時

- $RD(S)$ の中で第二項に a を含むものを全て削除する
- $\langle S, \mathtt{a}, \phi, \phi \rangle$を $RD(S)$ に加える．
- $RA(S)$ の中で，第二項に a を含む組が存在する時，a とエイリアスされている変数の RD も更新する．(2.4.2 節参照).

この文での SuDEF,PoDEF は，S で新たに生成される RD のうちの第二項と第四項のみを取り出したものであり，この文での MayALIAS,MustALIAS は，ϕである．

2.4.2 ポインタ変数が参照する領域への代入を行なう文および式

S : *p ← ···の形の時．ただし，$RA(S)$ の中に，代入される変数*p に関するもの $\langle S_{old}, (\mathtt{*p}, \mathtt{b}), (\mathtt{c}, \mathtt{d}) \rangle$が存在しているものとする．

- $RD(S)$ の中で第二項に*p を含むものを全て削除する.
- $RD(S)$ の中で第二項に b を，第三項に (*p,b) を，第四項に (c,d) を含むものを全て削除する．ただし，RA の第三項がϕの時には，RD の第二項，第三項との比較だけで良い.
- $RD(S)$ の中に，$\langle S_{old}, \mathtt{b}, \phi, \phi \rangle$の形のものがあれば，それも削除する．こうすることにより，[4] の解析における不正確さをなくすことができる.
- $\langle S, *\mathtt{p}, \phi, \phi \rangle$を $RD(S)$ に含める.
- $\langle S, \mathtt{b}, (*\mathtt{p}, \mathtt{b}), (\mathtt{c}, \mathtt{d}) \rangle$を $RD(S)$ に含める.

この文での SuDEF,PoDEF は，Sで新たに生成される RD の第二項と第四項のみを取り出したものであり，この文での MayALIAS,MustALIAS は，ϕである.

2.4.3 ポインタ変数を定義する文および式

S:p ← q の形の時

- $RA(S)$ から p の指す領域 (*p とする) を第二項に含むものを削除する.
- $RA(S)$ に $\langle S, (*\mathtt{p}, *\mathtt{q}), \phi \rangle$を追加する．さらに，すでに $RA(S)$ の中に*q に関するもの ($\langle S_{old}, (*\mathtt{q}, \mathtt{r}), (\mathtt{x}, \mathtt{y}) \rangle$ とする) があったとき，$RA(S)$ に $\langle S, (*\mathtt{p}, \mathtt{r}), (\mathtt{x}, \mathtt{y}) \rangle$ を追加する.

この文での SuDEF,PoDEF は，Sで新たに生成される RD の第二項と第四項のみを取り出したものであり，この文での MayALIAS,MustALIAS は，Sで新たに生成される RA の第二項と第三項のみを取り出したものである.

2.4.4 複合文

S:$S_1; S_2$の形のとき

複合文，条件文，繰り返し文における RD，RA の更新方法は，[1] のデータフロー方程式と同様であるので省略する．その他の集合は以下のように更新される.

$$SuDEF(S) = SuDEF(S_1) \cup SuDEF(S_2)$$
$$PoDEF(S) = PoDEF(S_1) \cup PoDEF(S_2)$$
$$MayALIAS(S) = MayALIAS(S_1) \cup MayALIAS(S_2)$$
$$MustALIAS(S) = MustALIAS(S_1) \cup MustALIAS(S_2)$$

2.4.5 条件文

S:**if** *expr* **then** S_1 **else** S_2 の形のとき

$$SuDEF(S) = SuDEF(expr) \cup (SuDEF(S_1) \cap SuDEF(S_2))$$
$$PoDEF(S) = PoDEF(expr) \cup PoDEF(S_1) \cup PoDEF(S_2)$$
$$MayALIAS(S) = MayALIAS(expr) \cup MayALIAS(S_1) \cup MayALIAS(S_2)$$
$$MustALIAS(S) = MustALIAS(expr) \cup (MustALIAS(S_1) \cap MustALIAS(S_2))$$

2.4.6 繰り返し文

S:**while** *expr* **do** S_1の形のとき

$$SuDEF(S) = SuDEF(expr)$$
$$PoDEF(S) = PoDEF(expr) \cup PoDEF(S_1)$$
$$MayALIAS(S) = MayALIAS(expr) \cup MayALIAS(S_1)$$
$$MustALIAS(S) = MustALIAS(expr)$$

2.4.7 手続き呼び出し

$S : P(exp_1, exp_2, \cdots)$ の形の時

1. 手続き Pの呼出側に，大域変数どうしのエイリアスの組が存在する場合，もしくは，Pのポインタ実引数が大域変数である場合それらの組からできるエイリアスが，すでに $preALIAS(P)$ に含まれていないかどうか調べる．含まれていれば，5. に進む.
2. 含まれていない時には，次に，Pが再帰呼び出しを行なっているかどうかを調べる．これは，手続き呼びだし情報を保持したスタックから容易に判断できる.
3. もし再帰呼び出しを行なっていれば，つまり，Pの呼び出しに関するものがすでにスタック上に存在すれば，この呼出文における各集合の変更は，すでに存在する Pの各集合を使って行なうことにし，5. へ

進む．このときの P の各集合の値は正確ではない可能性があるため，P は後でもう一度解析される（2.5 節参照）．

4. (a) 再帰呼び出しでない時には，S, $RD(S)$, $RA(S)$ をスタックに積み，$preALIAS(P)$ に $RA(S)$ に含まれるエイリアスを追加する．

   (b) RD および RA を以下のように初期化する．

$$
\begin{aligned}
RD(P_{entry}) &= RD(G_{in}) \cup RD(Para_{in}) \\
RA(P_{entry}) &= \langle P_{entry}, (*\mathbf{v}, \mathbf{v_{entity}}), \phi \rangle \\
&\quad \cup \{\langle P_{entry}, (\mathbf{pv_1}, \mathbf{pv_2}), \phi \rangle | \\
&\quad (\mathbf{pv_1}, \mathbf{pv_2} \in preALIAS(P)\}
\end{aligned}
$$

   ここで，P_{entry} は，手続きの入口を示し，$(*\mathbf{v}, \mathbf{v_{entity}})$ は，ポインタ型実引数 $\mathbf{v}$ の指す領域と，それに対応する手続き引数の仮の領域 $\mathbf{v_{entity}}$ とのエイリアスを示す．

   (c) 手続き P を解析する．

   (d) 解析後の SuDEF, PoDEF, MayALIAS, MustALIAS から，$SDglob(P)$, $PDglob(P)$, $SDpara(P)$, $PDpara(P)$, $MayALIAS(P)$, $MustALIAS(P)$ を更新する．$SDglob(P)$, $PDglob(P)$ は，それぞれ P の解析終了後の SuDEF, PoDEF のうち大域変数およびそれが指す領域に関するものを集めたものであり，$SDpara(P)$, $PDpara(P)$ は SuDEF, PoDEF のうち引数およびそれが指す領域 (v_{entity}) に関するものを集めたものである．また，$MayALIAS(P)$, $MustALIAS(P)$ には，解析によって得られた MayALIAS, MustALIAS のうち，大域変数もしくは引数に関する組のみを含んでいるものを集める．

   (e) スタックから S, $RD(S)$, $RA(S)$ を pop する．

5. (a) 各集合を以下のように設定する．

$$
\begin{aligned}
SuDEF(S) &= SDglob(P) \cup SDpara(P) \\
PoDEF(S) &= PDglob(P) \cup PDpara(P) \\
MayALIAS(S) &= MayALIAS(P) \\
MustALIAS(S) &= MustALIAS(P)
\end{aligned}
$$

   ただし，このとき，手続きの各集合の全ての要素を含むのではなく，$RA(S)$ に含まれる要素の第二項に関するもののみを含むようにする．

   (b) RD から，$SDglob(P)$, $SDpara(P)$ に対応している変数に含まれるものを削除する．そして，$PDglob(P)$, $PDpara(P)$ に含まれているものを追加する．ただしこの時，対象となる手続きの各集合の要素は，その第二項に，ϕ か，preALIAS に含まれるエイリアスのみの組で構成されるものを含んでいるものである．また，手続きの解析の時に用意した仮の領域を表す変数を，各々の実引数に対応させて調べながら更新を行なう．この時追加される RD の第一項，すなわちこの RD が生成される文番号は，S および，PDG 上での対応する変数の *g-out* もしくは *para-out* 節点である（3 節参照）．また，第三項，第四項は，削除された RD のうち，それに対応するものが持っていたものと同じものとする．

   (c) $RA(S)$ のうち，$MustALIAS(P)$ に含まれている変数を含む組から，その変数を削除する．そして，$MayALIAS(P)$ を追加する．追加される RA の第一項は，S とする．また，第三項は，削除された RA のうち，それに対応するものが持っていたものと同じものとする．

2.5 再帰呼び出しの処理

再帰呼び出しを行なっている手続きを解析する場合，呼出文が現れた時点で，そのままスタックに積むだけでは正確な解析が行なえない．

そこで，再帰呼び出しを行なっている手続きを解析する場合には，[6] のように，2.1 節で定義した手続きの各集合が変化しなくなるまで解析を行ない続けるという方法をとる．

3 PDG の生成

2 節で述べたアルゴリズムでプログラムを解析することにより，プログラム中の各文の依存関係がわかる．それを用いて，プログラム依存グラフ（Program Dependence Graph, 略して PDG）を生成できる．

PDG の各ノードは，プログラム中の文または条件節および中継節点を表す．中継節点とは，プログラム中の文とは直接対応しないが，手続き境界を越える依存を伝えるために，特別に用意したものである．これらの中継節点は，[6] において用いられている特殊節点を拡張したものである．表 1 にその一覧を示す．

PDG のアークは，データ依存関係辺と制御依存関係辺の二種類がある．データ依存関係辺は，変数の定義-参照の関係を表す．データ依存関係辺には，それが伝える変数の定義に関する変数名およびそれが存在するためのエイリアスがラベルづけされる．これは RD および RA から知ることができる．

表 1: 中継節点一覧

exit 節点	関数の戻り値の影響を呼び出した節点に伝えるための節点で，各関数に一つずつある．
g–in 節点	手続き外からの大域変数の影響を伝えるための節点で，各手続きごとに，その中で参照される大域変数それぞれに対して一つずつある．
g–out 節点	手続きの中で定義された大域変数の影響を呼出側に伝えるための節点で，各手続きごとに，その中で定義される大域変数それぞれに対して一つずつある．
para–in 節点	手続きの引数を通して伝わる影響を検出するための節点で，その手続きの引数それぞれに対して一つずつある．
para–out 節点	手続きの引数が参照渡しで呼ばれた時，手続き内部で引数が定義された時の影響を外部に伝えるための節点である．

制御依存関係辺は，条件文や繰り返し文の条件判定部分から，そこに含まれるブロック内の全ての文に引かれる．これはプログラムの構造から容易に知ることができる．

4 スライシング

スライシング [5] とは，プログラム中のある文で参照もしくは定義される変数に関係のある部分だけをプログラムから抽出する技術であり，抽出された部分をスライスと呼ぶ．

スライシングは，PDG の辺を辿ることによって行なうことができる [6]．ただし，データ依存関係辺を辿る場合には，その辺にラベルづけされているエイリアスに注意する必要がある．

PDG を辿る段階で，あるノードに到達した時，次に辿る辺は，それまで辿ってきた辺にラベル付けされていたエイリアスと同じエイリアスでラベル付けされた辺か，そのラベルがϕであるような辺である．ただし，ラベルがϕであるような辺を辿ってきた場合には，このような比較は行なわずに，全ての辺を辿るものとする．

従って，辿った先のノードがすでに辿られているからといってそこで探索をやめてはならない．そのかわりに，辺がすでに辿られていればそこでやめてもよい．

そのようにして PDG を辿った結果，辿られたノードの集合がスライスとなる．

5 適用例

本アルゴリズムの適用例として，図 1 のプログラムを解析することを考える．

この解析結果は，表 2 のようになる．

ここで，$\mathbf{c}_{\mathbf{entity}}$,$\mathbf{d}_{\mathbf{entity}}$は，それぞれ手続き proc における仮引数 c,d の指す仮の領域を表す．これらの領域は，手続き呼び出しの副作用を正確に解析するために必要となる [2]．S17 における，手続き proc の呼びだしの結果，2.1 節で定義された各集合の値は以下のようになる．

```
S1: int b;
S2: int proc(int *c,*d)
S3: {
S4:   int *t = c;
S5:   if(*d == 1){
S6:     c = d;
S7:     d = t;
S8:     *c = b;
S9:   }else{
S10:    *d = 3;
S11:  }
S12: }

S13: void main()
S14: {
S15:   int a = 2;
S16:   b = 1;
S17:   proc(&a,&b);
S18:   proc(&a,&b);
S19:   printf("%d %d",a,b);
S20: }
```

図 1: サンプルプログラム

$$
\begin{aligned}
SDglob(\mathbf{proc}) &= \{\langle \mathbf{b}, (\mathbf{d}_{\mathbf{entity}}, \mathbf{b})\rangle\} \\
PDglob(\mathbf{proc}) &= \{\langle \mathbf{b}, (\mathbf{d}_{\mathbf{entity}}, \mathbf{b})\rangle\} \\
ImUSE(\mathbf{proc}) &= \{\langle \mathbf{b}, \phi\rangle, \langle \mathbf{b}, (\mathbf{d}_{\mathbf{entity}}, \mathbf{b})\rangle\} \\
SDpara(\mathbf{proc}) &= \{\langle \mathbf{d}_{\mathbf{entity}}, \phi\rangle\} \\
PDpara(\mathbf{proc}) &= \{\langle \mathbf{d}_{\mathbf{entity}}, \phi\rangle\} \\
MustALIAS(\mathbf{proc}) &= \phi \\
MayALIAS(\mathbf{proc}) &= \phi \\
preALIAS(\mathbf{proc}) &= \{(\mathbf{b}, \mathbf{d}_{\mathbf{entity}})\}
\end{aligned}
$$

S18 での手続き proc の二回目の呼び出しの時には，その時点で存在するエイリアスがすでに proc の preALIAS に含まれているので，proc を再解析することなく $RD(S18)$ および $RA(S18)$ を更新している．

この解析結果に基づき生成された PDG を 図 2 に示す．

表 2: 解析結果

文番号	RD	RA
S15	$\langle S15, \mathtt{a}, \phi, \phi \rangle$	ϕ
S16	$\langle S15, \mathtt{a}, \phi, \phi \rangle$, $\langle S16, \mathtt{b}, \phi, \phi \rangle$	ϕ
P_{entry}	$\langle \mathtt{c_{entity}} - \mathrm{in}, \mathtt{c_{entity}}, \phi, \phi \rangle$, $\langle \mathtt{d_{entity}} - \mathrm{in}, \mathtt{d_{entity}}, \phi, \phi \rangle$, $\langle \mathtt{b} - \mathtt{g} - \mathrm{in}, \mathtt{b}, \phi, \phi \rangle$	$\langle P_{entry}, (*\mathtt{c}, \mathtt{c_{entity}}), \phi \rangle$, $\langle P_{entry}, (*\mathtt{d}, \mathtt{d_{entity}}), \phi \rangle$, $\langle P_{entry}, (\mathtt{b}, \mathtt{d_{entity}}), (\mathtt{d_{entity}}, \mathtt{b}) \rangle$, $\langle P_{entry}, (\mathtt{b}, *\mathtt{d}), (\mathtt{d_{entity}}, \mathtt{b}) \rangle$
S4	$RD(P_{entry})$	$RA(P_{entry})$, $\langle S4, (*\mathtt{t}, *\mathtt{c}), \phi \rangle$, $\langle S4, (*\mathtt{t}, \mathtt{c_{entity}}), \phi \rangle$
S5	$RD(S4)$	$RA(S4)$
S6	$RD(S5)$	$\langle P_{entry}, (\mathtt{b}, \mathtt{d_{entity}}), (\mathtt{d_{entity}}, \mathtt{b}) \rangle$, $\langle P_{entry}, (*\mathtt{d}, \mathtt{d_{entity}}), \phi \rangle$, $\langle P_{entry}, (*\mathtt{d}, \mathtt{b}), (\mathtt{d_{entity}}, \mathtt{b}) \rangle$, $\langle S4, (*\mathtt{t}, \mathtt{c_{entity}}), \phi \rangle$, $\langle S6, (*\mathtt{c}, *\mathtt{d}), \phi \rangle$, $\langle S6, (*\mathtt{c}, \mathtt{d_{entity}}), \phi \rangle$, $\langle S6, (*\mathtt{c}, \mathtt{b}), (\mathtt{d_{entity}}, \mathtt{b}) \rangle$
S7	$RD(S6)$	$\langle P_{entry}, (\mathtt{b}, \mathtt{d_{entity}}), (\mathtt{d_{entity}}, \mathtt{b}) \rangle$, $\langle S4, (*\mathtt{t}, \mathtt{c_{entity}}), \phi \rangle$, $\langle S6, (*\mathtt{c}, \mathtt{d_{entity}}), \phi \rangle$, $\langle S6, (*\mathtt{c}, \mathtt{b}), (\mathtt{d_{entity}}, \mathtt{b}) \rangle$ $\langle S7, (*\mathtt{d}, *\mathtt{t}), \phi \rangle$, $\langle S7, (*\mathtt{d}, \mathtt{c_{entity}}), \phi \rangle$,
S8	$\langle S8, *\mathtt{c}, \phi, \phi \rangle$, $\langle S8, \mathtt{d_{entity}}, (*\mathtt{c}, \mathtt{d_{entity}}), \phi \rangle$, $\langle S8, \mathtt{b}, (*\mathtt{c}, \mathtt{b}), (\mathtt{d_{entity}}, \mathtt{b}) \rangle$, $RD(P_{entry})$	RA(S7)
S10	$\langle S10, *\mathtt{d}, \phi, \phi \rangle$, $\langle S10, \mathtt{d_{entity}}, (*\mathtt{d}, \mathtt{d_{entity}}), \phi \rangle$, $\langle S10, \mathtt{b}, (\mathtt{d_{entity}}, \mathtt{b}), (\mathtt{d_{entity}}, \mathtt{b}) \rangle$, $\langle S10, \mathtt{b}, (*\mathtt{d}, \mathtt{b}), (\mathtt{d_{entity}}, \mathtt{b}) \rangle$, $RD(P_{entry})$	RA (S4)
P_{exit}	$RD(S8) \cup RD(S10)$	$RA(S4) \cup RA(S8)$
S17	$\langle S16, \mathtt{a}, \phi, \phi \rangle$, $\langle S17, \mathtt{b}, \phi, \phi \rangle$, $\langle \mathtt{b} - \mathtt{g} - \mathrm{out}, \mathtt{b}, \phi, \phi \rangle$, $\langle \mathtt{d_{entity}} - \mathrm{out}, \mathtt{b}, \phi, \phi \rangle$	ϕ
S18,S19	$\langle S16, \mathtt{a}, \phi, \phi \rangle$, $\langle S18, \mathtt{b}, \phi, \phi \rangle$, $\langle \mathtt{b} - \mathtt{g} - \mathrm{out}, \mathtt{b}, \phi, \phi \rangle$, $\langle \mathtt{d_{entity}} - \mathrm{out}, \mathtt{b}, \phi, \phi \rangle$	ϕ

このうち，楕円が中継節点で，四角が通常の節点である．また，辺のうち，実線のものがデータ依存関係辺で，破線のものが制御依存関係辺である．

次に，作成されたPDGからスライシングを行なう．図1におけるS19での出力文で参照される変数aに関する(後方)スライスを表したのが図3である．変数aに影響を与えない，手続きprocやその呼び出し文などはスライスに含まれていないことがわかる．

6 関連研究

これまでにも，ポインタ変数やエイリアスを含むプログラムの解析アルゴリズムはいくつか発表されている．

[3]では，Conditional May Alias informationに基づいて，C言語などの，ポインタ変数によるエイリアスの副作用が手続きを越えて伝わるようなプログラムの解析を行なっている．しかし，このアルゴリズムは得られるエイリアスの正確さが不十分であった．

[4]では，[3]を改良したアルゴリズムを用いて，C言語で書かれたプログラムの依存関係解析を行なっている．このアルゴリズムでは手続き呼び出しがある度に呼び出された手続きの解析をやり直している．本研究で提案するアルゴリズムでは，手続き呼びだし地点でのエイリアスを調べることにより，手続きを再解析し直すかどうかを決定しながら解析を行なっているため，効率がよい．また，[4]のアルゴリズムでは，解析結果のRDの中に不正確な要素が含まれてしまう場合があるが，これについても，本研究で提案するアルゴリズムでは，独自のRD解

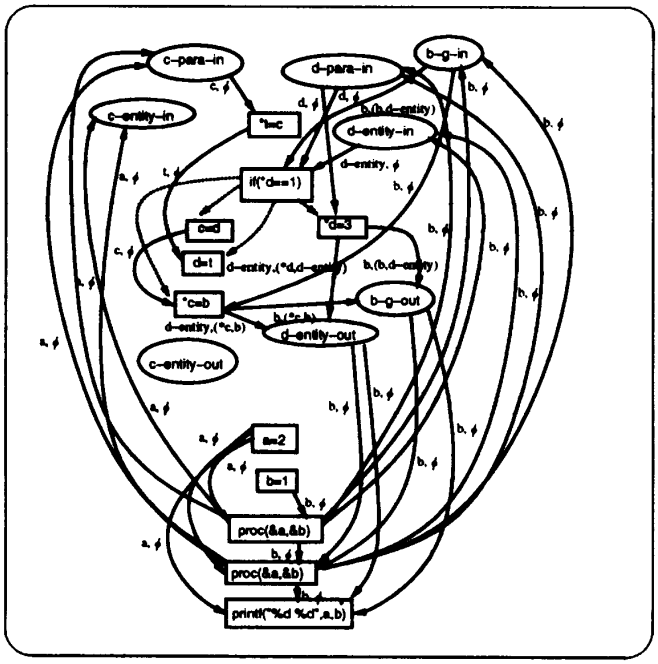

図 2: PDG

```
 S1: int b;

S13: void main()
S14: {
S15:   int a = 2;

S19:   printf("%d %d",a,b);
S20: }
```

図 3: スライス

析手法を用いることにより，そのような不正確な要素を含まないようにしている．

[2] では，[3] の不正確性を改善し，さらに時間的にも効率の良いエイリアス解析アルゴリズムを提案している．このアルゴリズムでは，まず各手続きの全ての呼びだし地点において静的に求められるエイリアスを全て求めておき，そのエイリアスのもとで各手続きを，手続きの呼出順序を考慮して解析するという手法をとっている．これは，一見効率が良いように見えるが，例えば，一番最初に解析される手続き P(どの手続きも呼ばない手続き) の呼びだし地点のエイリアスは，P の後で解析される手続きを解析した後に変わってしまう場合がある．そのような場合には，また解析をやり直さなければならない．本研究で提案するアルゴリズムでは，このような場合を考えて主手続きからの解析を行なっているため，このような解析のやり直しは起こらない．

7 まとめ

手続きごとに副作用の情報を保持しておくことにより，一つの手続きが複数回解析されるような場合に，再解析することなく RD や RA を更新することで効率の良い依存関係解析を行なえるアルゴリズムを提案した．

今後の課題としては，アルゴリズムの検証や，プロトタイプの実装を行なって，実際の実行効率を評価することなどが挙げられる．

参考文献

[1] Alfred V. Aho, Ravi Sethi, and Jefferey D. Ullman. *Compilers:Principles, Techniques, and Tools.* Addison Wesley, 1986.

[2] Jong-Deok Choi, Michael Burke, and Paul Carini. Efficient flow-sensitive interprocedural computation of pointer-induced aliases and side effects. In *Conference Record of 20th ACM Symposium on Principles of Programming Languages*, pages 232–245, Charleston, South Carolina, January 1993.

[3] William Landi and Barbara G. Ryder. A safe approximate algorithm for interprocedural pointer aliasing. In *Proceedings of the ACM SIGPLAN '92 Conference on Programming Languages Design and Implementation*, pages 235–248, San Francisco, California, June 1992.

[4] Hemant D. Pande, William A. Landi, and Barbara G. Ryder. Interprocedural def-use associations for c systems with single level pointers. *IEEE Transanctions on Software Engineering*, 20(5):385–402, May 1994.

[5] Mark Weiser. Program slicing. In *Proceedings of the Fifth International Conference on Software Engineering*, pages 439–449, San Diego, CA, September 1981.

[6] 植田 良一, 練 林, 井上 克郎, 鳥居 宏次. 再帰を含むプログラムのスライス計算法. 電子情報通信学会論文誌, J78-D-I(1):11–22, 1995-1.