論　文

エイリアスフローグラフを用いたオブジェクト指向プログラムのエイリアス解析手法

大畑　文明[†]　　近藤　和弘[†]　　井上　克郎[†,††]

Alias Analysis Method for Object Oriented Programs Using Alias Flow Graphs

Fumiaki OHATA[†], Kazuhiro KONDOU[†], and Katsuro INOUE[†,††]

あらまし　プログラムテキスト上の式（または部分式）の対が同一のオブジェクト（メモリ領域）を指す場合，それらの式はエイリアス関係にあるという．これらは，引き数の参照渡し，参照変数，ポインタを介した間接参照などで生じる．既存のエイリアス解析手法は，解析結果の再利用が不十分で，オブジェクト指向プログラムのもつ再利用性がエイリアス解析に生かされていない．また，エイリアス解析手法の提案は今までいろいろなされているが，実用的なツールはあまり知られていない．本研究では，エイリアス解析結果の再利用及びモジュール化を考慮した，オブジェクト指向プログラムに対するエイリアス解析手法を提案する．本手法を実現したJAVAエイリアス解析ツールは，JDK（JAVA Developers's Kit）付属クラスライブラリなどの大規模プログラムを実用的な時間で解析することができる．

キーワード　エイリアス，モジュール化，オブジェクト指向プログラム，JAVA

1. まえがき

プログラムテキスト上の式（または部分式）の対が同一のオブジェクト（メモリ領域）を指す場合，それらの式はエイリアス関係（Alias Relation）にあるという．エイリアス関係は，引き数の参照渡し，参照変数，ポインタを介した間接参照などによって生じる．エイリアス関係は同値関係であり，その同値類をエイリアス集合（Alias Set）と呼ぶ．また，プログラムを静的に解析しエイリアス集合を求めることをエイリアス解析（Alias Analysis）といい，コンパイル時の最適化 [1] や，プログラムスライス（Prgram Slice）[13] の計算に欠くことのできないものである．

また，現在のプログラム開発環境において，C などの手続き型言語だけでなく，JAVA [7]，C++ [3] 等いわゆるオブジェクト指向言語の利用が高まっている．オブジェクト指向言語には，従来の手続き型言語にはないクラス，継承，動的束縛，ポリモルフィズムなど新しい概念が導入されており，それらに対応したエイリアス解析手法が提案されている [14].

しかし，既存のエイリアス解析手法は，解析手法をオブジェクト指向言語に対応させたにすぎず，解析結果そのものの再利用性，モジュール性は満たされていない．プログラムの大規模化，プログラムの再利用への関心が高まる現在，これらの特性を満たすエイリアス解析手法が望まれる．

本研究では，再利用性，モジュール性を考慮したエイリアスフローグラフによるエイリアス解析手法の提案を行う．また，提案手法をJAVA エイリアス解析ツールとして実装しその有効性を評価する．本ツールは，近年ソフトウェア開発言語として多く利用されているJAVA を対象としており，JDK（JAVA Developers's Kit）付属クラスライブラリなどの大規模プログラムに対しても実用的な時間で解析することができる．

以降，**2.** ではエイリアス解析及びオブジェクト指向プログラムに対する解析について説明する．**3.** で手法の提案を行う．**4.** でJAVA エイリアス解析ツールによる手法の実験的評価を行い，**5.** で考察する．最後に **6.** でまとめと今後の課題について述べる．

† 大阪大学大学院基礎工学研究科，豊中市
Graduate School of Engineering Science, Osaka University, Toyonaka-shi, 560–8531 Japan
†† 奈良先端科学技術大学院大学情報科学研究科，生駒市
Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma-shi, 630–0101 Japan

2. オブジェクト指向プログラムにおけるエイリアス解析

2.1 エイリアス解析

エイリアス解析は，大きくFIエイリアス解析（Flow-Insensitive Alias Analysis）（以降，FI解析と略す），FSエイリアス解析（Flow-Sensitive Alias Analysis）（以降，FS解析と略す）の二つに分けることができる．以下，それぞれを図1を用いて簡単に説明する．

[Flow-Insensitive（FI）エイリアス解析]

FIエイリアス解析[2], [12]とは，各プログラム文の実行順を考慮しないエイリアス解析手法をいい，エイリアスグラフを利用する．図1(a)に変数c（太枠部）のFIエイリアス（網掛部）を，図1(c)にその計算に用いたエイリアスグラフを示す．エイリアスグラフは無向グラフであり，節点はメモリ領域を指し得る変数及び式を，辺は（代入や引き数の参照渡しなどにより）節点間に直接のエイリアス関係があることを表す．文7の変数cに関するエイリアスを求める場合，エイリアスグラフの変数c節点から到達可能な節点は{a, b, **new** Integer(1), **new** Integer(2))}であり，網掛部が求めるエイリアスとなる．

[Flow-Sensitive（FS）エイリアス解析]

FSエイリアス解析[9], [16]とは，プログラム文の実行順を考慮したエイリアス解析手法をいい，到達エイリアス集合（Reaching Alias Set）を利用する．図1(b)に変数c（太枠部）のFSエイリアス（網掛部）を，図1(d)にその計算に用いた到達エイリアス集合を示す．到達エイリアス集合$RA(s)$の要素は文sの実行直前において成立しかつ文sにおいて識別子を介して参照可能なエイリアス集合であり，その集合の各要素は（文番号，式）の組で表される．文7の変数cに関するエイリアスを求める場合，$RA(7)$から識別子cを含むエイリアス集合を探索する．変数cは集合{(6, c), (2, a), (6, a), (2, **new** Integer(1))}に含まれており，網掛部が求めるエイリアスとなる．

[FIエイリアス解析とFSエイリアス解析]

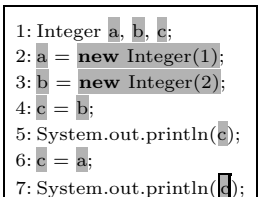
FS解析はプログラム文の実行順を考慮しているため，FI解析と比較し時間計算量，空間計算量を必要とするが，解析の精度は高く，実際に図1においてもその差は顕著である．両者の詳細な比較は[11]でなされており，本研究では解析精度を重視していることからFS解析に着目する．

2.2 オブジェクト指向プログラムにおけるエイリアス解析

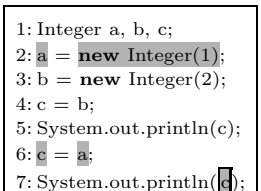
既存のオブジェクト指向プログラムにおけるエイリアス解析手法は，既に提案されている手続き型言語のエイリアス解析手法をオブジェクト指向言語に拡張したものとなっているが，それらに関していくつかの視点から考察を行う．

[FS解析の問題]

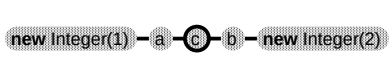
手続きを含むプログラムを解析する場合，到達エイリアス集合を用いたFS解析では，すべての実行（手続き呼出し）経路をたどりながら到達エイリアス集合を計算しなければならない．また，再帰呼出しが存在する場合，エイリアス集合が収束するまで再帰経路を解析し続ける必要がある．これは，解析結果がエイリアス集合の形で保持されていることに起因している．つまり，ある時点でエイリアス集合に変化が起こったとき，その変化の影響を受ける可能性のある到達エイ



(a) FI エイリアス解析



(b) FS エイリアス解析



(c) エイリアスグラフ

文 (s)	到達エイリアス集合 ($RA(s)$)
1	ϕ
2	ϕ
3	{(2, a), (2, **new** Integer(1))}
4	{(2, a), (2, **new** Integer(1))}, {(3, b), (3, **new** Integer(2))}
5	{(2, a), (2, **new** Integer(1))}, {(4, c), (3, b), (4, b), (3, **new** Integer(2))}
6	{(2, a), (2, **new** Integer(1))}, {(4, c), (5, c), (3, b), (4, b), (3, **new** Integer(2))}
7	{(6, c), (2, a), (6, a), (2, **new** Integer(1))}, {(3, b), (4, b), (3, **new** Integer(2))}

(d) 到達エイリアス集合

図1 FIエイリアス解析とFSエイリアス解析
Fig. 1 FI alias analysis and FS alias analysis.

リアス集合をもつ文すべてにその変化を伝搬させ，各文の到達エイリアス集合を再計算しなければならないためである．

[解析結果の再利用]

到達エイリアス集合 $RA(s)$ は文 s が存在するプログラム全体を解析することにより導出されたものであるため，別の文 t が変更されたとき，$RA(s)$ を再計算しなければならない場合が多く存在する．これは，エイリアス解析結果のモジュール性，独立性が満されていないことに起因する．オブジェクト指向プログラムでは，継承機能など，言語自身が再利用を考慮したものとなっているため，記述されたクラスの利用範囲が特定のプログラムにとどまらない．また，上位クラスであるほどその属性やメソッドは汎化されたものとなり，一度定義されると変更されることは少ない．そのため，解析結果のクラスやメソッド単位でのモジュール化は，再計算コストの削減に有効であると考えている．

[同一クラスの異なるインスタンス属性の取扱い]

オブジェクト指向プログラムでは，異なるオブジェクト間での状態（属性）と振舞い（メソッド）は独立であるため，図 2 の例では {x.s, A::s}，{y.s, A::s} がそれぞれエイリアス集合といえるが，同一クラスのインスタンスがその内部情報を共有する場合，{x.s, y.s, A::s} がエイリアス集合とみなされるため，解析精度の低下につながる．このため，インスタンスごとにクラス内部のエイリアス情報をもたせる手法が考えられるが，単純にインスタンスの数だけエイリアス情報を生成すると多大な空間コストを要する．そこで，存在するエイリアス関係を

- 内部エイリアス関係（Inner Alias Relation）同一クラスのインスタンス間で共通なエイリアス関係
- 外部エイリアス関係（Outer Alias Relation）同一クラスのインスタンス間で独立なエイリアス関係

に分け，内部エイリアス関係のみ前もって解析し，外部エイリアス関係は必要に応じて逐次導出する手法が考えられる．

```
x = new A;
y = new A;
x.s = ...;
y.s = ...;
```

```
class A {
  Integer s;
  ...(...) {
    ... = s;
```

図 2　オブジェクト指向プログラム（Java）
Fig. 2　Object oriented program (Java).

3. エイリアスフローグラフ（AFG）を用いたエイリアス解析手法

本研究では，前章で述べた再解析コストの軽減，エイリアス解析結果のモジュール化，内部エイリアス関係及び外部エイリアス関係の抽出のための，

（1） エイリアスフローグラフ（AFG）構築法

（2） AFG によるエイリアス計算手法

の提案を行う．エイリアスフローグラフ（Alias Flow Graph，AFG）は，FS エイリアス関係を無向グラフで表現したものであり，FS エイリアスの計算をグラフの到達問題に置き換える．本章で述べるアルゴリズムはJava を前提に，参照型の式（参照変数，インスタンス生成式など）によるエイリアスを対象としているが，ポインタ変数の間接参照によるエイリアスにも拡張可能であり，**5.2** で簡単に述べる．

AFG によるエイリアス解析手法は以下四つのフェーズで構成される．

Phase 1：　AFG 節点（オブジェクトへの参照）の抽出
Phase 2：　AFG 辺（直接のエイリアス関係）の抽出
Phase 3：　メソッド呼出しグラフの構築
Phase 4：　エイリアス計算

クラス，メソッド単位で到達エイリアス集合に基づく FS エイリアス解析を行い，クラス AFG（Class AFG），メソッド AFG（Method AFG）をそれぞれ構築する（Phase 1〜2）．各 AFG は独立したモジュールとして存在し，対応するクラス，メソッドが更新されない限り不変であり，2 次媒体への記憶及びその再利用が可能である．AFG は内部エイリアス関係のみ保持しており，外部エイリアス関係はユーザからの要求があったとき逐次解析する（Phase 4）．

以降，メソッドのない単一プログラム（図 3），複数メソッドで構成されたプログラム（図 8）を例に用いながら各フェーズを順に説明する．

3.1　Phase 1：AFG 節点の抽出

AFG 節点（AFG Node）は，文番号とオブジェクトへの参照式の組である．ここでいうオブジェクトへの参照式とは，参照型の式を指し，参照変数及びインスタンス生成式もこれに含まれる．また，これらの節点を特に AFG 標準節点（AFG Normal Node）と呼ぶ．

図 3 の文番号 3 の b は参照変数であるためオブジェクトへの参照式に該当し，AFG 節点 (3, b) が生

表 1 特殊節点
Table 1 AFG special node.

特殊節点	概要
Actual-Alias-in（AA-in）	実エイリアス引数（実引数により，メソッドに渡されるエイリアス）
Formal-Alias-in（FA-in）	仮エイリアス引数（仮引数により，メソッドに渡されるエイリアス）
Actual-Alias-out（AA-out）	実エイリアス引数（実引数により，メソッド呼出し元に渡されるエイリアス）
Formal-Alias-out（FA-out）	仮エイリアス引数（仮引数により，メソッド呼出し元に渡されるエイリアス）
Method-Alias-out（MA-out）	戻りエイリアス値（戻り値により，メソッド呼出し元に渡されるエイリアス）
Method	メソッド呼出し（メソッドの戻り値により，メソッド呼出し元に渡されるエイリアス，AA-in（out）節点の親節点）
Instance-Alias-in（IA-in）	エイリアス属性（属性により，メソッド内に渡されるエイリアス）
Instance-Alias-out（IA-out）	エイリアス属性（属性により，メソッド外に渡されるエイリアス）

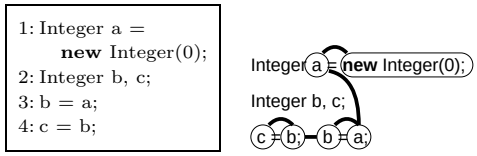
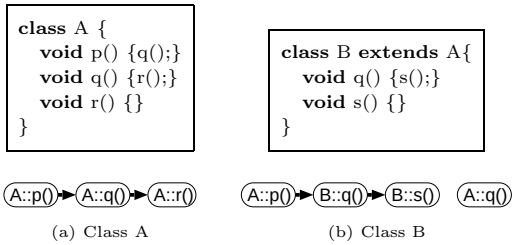


図 3 サンプルプログラム及びその AFG
Fig. 3 Sample program and its AFG.



(a) Class A (b) Class B

図 4 サンプルプログラム及びその MFG
Fig. 4 Sample program and its MFG.

成される(注1)．同様に，文番号 1 の**new** Integer(0) はインスタンス生成式であるため，AFG 節点 (1, **new** Integer(0)) が生成される．

メソッドのない単一プログラムにおいては，AFG 標準節点のみ用いることで AFG を構築できる．一方，通常オブジェクト指向プログラムではメソッド引数やインスタンス（クラス）属性が存在するため，これらを介したエイリアスの受渡しを考慮しなければならない．そのため，表 1 に挙げる AFG 特殊節点（AFG Special Node）を追加する．

3.2 Phase 2：AFG 辺の抽出

AFG 標準辺（AFG Normal Edge）は，二つの AFG 節点間で，同一変数の参照や代入文，パラメータの対応などによる直接のエイリアス関係が存在するとき引かれる．複数の AFG 辺で構成された経路は，2 節点間の（推移的に成り立つ）間接のエイリアス関係を表す．

また，AFG をオブジェクト指向プログラムに適応する際，*a.b* や *c.d*() のようなインスタンス属性やインスタンスメソッドを表現するために，インスタンス及び属性（メソッド）を表す AFG 節点間に親子関係を定義する．例えば *a.b* に関して，*a*，*b* に対応する AFG 節点をそれぞれ N_a，N_b とすると，N_a は N_b の親節点（Parent Node），N_b は N_a の子節点（Child Node）となり，N_a から N_b に対し有向辺が引かれる（このような辺を AFG 特殊辺（AFG Special Edge）と呼ぶ）．図 9 にその具体例（破線の有向辺）を示す．この関係はエイリアス計算時（Phase 4）に利用される．このような節点間の親子関係は Method 節点と AA-in（out）節点間にも存在し，これらに対しても同様に AFG 特殊辺を引く．

3.3 Phase 3：メソッド呼出しグラフの構築

メソッド呼出しグラフ（Method Flow Graph, MFG）とは，クラス中に存在する（同一インスタンス）メソッド間の呼出し関係を有向グラフで表現したものである．MFG の各節点を MFG 節点（MFG Node）と呼び，メソッドを表す．メソッド A がメソッド B を呼ぶ場合，A，B に対応する MFG 節点をそれぞれ M_A，M_B とすると，有向辺である MFG 辺（MFG Edge）が節点 M_A から節点 M_B に引かれる．

図 4 にサンプルプログラム及びその MFG を示す．B クラスではメソッド p は定義されておらず，シグニチャp() に対するメソッドは A::p() となる．B クラスの MFG 構築の際には，B クラスが継承したメソッド A::p() はシグニチャq() のメソッドを呼び出しているが，それは B::q() でありメソッド A::q() ではないことに留意する必要がある．

(注1)：本論文に図示されている AFG に関して，AFG 節点のラベルには文番号が省略されているが，わかりやすいようプログラムテキストの文字列をグラフ上に補助的に記している．

Step 1: $E := \varphi(e)$
Step 2: E を始点とし，新たな到達可能節点が見つからなくなるまで AFG 辺を推移的にたどる
加えて，到達節点 C の種類に応じて以下のような処理も行う
[Cond.1] 親節点をもつ標準（インスタンス属性）節点：
（1） $c := \varphi^{-1}(C)$, $P := N_P(C)$
（2） $\{G_M \mid \psi^{-1}(G_M) \in$ Object-Context($\mathcal{P}$)$\}$ に含まれる IA-in（out）[c] 節点から AFG 辺をたどる
$\{N \mid N_P(N) \in \mathcal{P}$ && $\varphi^{-1}(N) =_{id} c\}$ から AFG 辺をたどる
[Cond.2] 親節点をもつ Method 節点：
（1） $c := \varphi^{-1}(C)$, $P := N_P(C)$
（2） $\{M \mid \psi^{-1}(M) \in$ Object-Context($\mathcal{P}$)$\}$ に含まれる MA-out 節点から AFG 辺をたどる
[Cond.3] IA-in（out）節点：
（1） $c := \varphi^{-1}(C)$
（2） $\{M \mid \psi^{-1}(M) \in$ Object-Context(**this**)$\}$ に含まれる IA-out（in）[c] 節点から AFG 辺をたどる
[Cond.4] AA-in（out）節点：
（1） Method 節点 $N_P(C)$ が呼ぶメソッド m を計算
（2） $\psi(m)$ に含まれる FA-in（out）節点から AFG 辺をたどる
[Cond.5] FA-in（out）節点：
（1） $M_B := \{M \mid C \in M\}$, $m_B := \psi^{-1}(M_B)$
（2） $m_A := \{m \mid m \in$ Object-Context(**this**) && m が m_B を呼ぶ $\}$（MFG より），$M_A := \psi(m_A)$
（3） M_A に含まれる AA-in（out）節点から AFG 辺をたどる
[Cond.6] MA-out 節点：
（1） $M_B := \{M \mid C \in M\}$, $m_B := \psi^{-1}(M_B)$
（2） $m_A := \{m \mid m$ が m_B を呼ぶ $\}$（MFG より），$M_A := \psi(m_A)$
（3） M_A に含まれる（m_B を呼ぶ）Method 節点から AFG 辺をたどる
[Cond.7] Method 節点：
（1） 節点 C に対応するメソッド呼出し文から呼ばれるメソッドを m_A とする
（2） $M_A := \psi(m_A)$
（3） M_A に含まれる MA-out 節点から AFG 辺をたどる
Step 3: 到達可能な AFG 節点集合に対応するオブジェクトへの参照式が，求める e のエイリアス

図 5　エイリアス計算アルゴリズム
Fig. 5　Alias calculation algorithm.

3.4　Phase 4：エイリアス計算

AFG によるエイリアス計算は以下の方針に基づく．

（1） 親節点 P をもつ節点 C のエイリアスを導出する場合，まず P のエイリアス計算問題を解決させ（これにより，エイリアス解析対象が特定インスタンスに限定され，P の型及び後述するオブジェクトコンテクストが導出される），その結果をもとに C のエイリアス計算を行う

（2） MA-out や FA-in（out）節点など，メソッド間をまたぐエイリアス関係の導出の際には，MFG を用いて呼出し先（呼出し元）メソッドを探索し，対応する Method 節点や AA-in（out）節点から AFG 辺をたどる

アルゴリズムの詳細は図 5 に，またその中で利用する式の定義を図 6 に示す．

[オブジェクトコンテクスト]

AFG を用いて導出されたエイリアス集合 $\mathcal{P}$ に関するオブジェクトコンテクスト（Object Context）とは，$\mathcal{P}$ に属する節点の子節点として存在する Method 節点から導出される，$\mathcal{P}$ が直接若しくは間接的に呼び出す可能性のあるメソッド集合を指し，Object-Context($\mathcal{P}$) と表す．

$\varphi(x)$:　式 x から AFG 節点 X への射影
$\mathcal{X}$:　X のエイリアス（$\mathcal{X}$ は AFG 節点の集合）
$N_P(X)$:　X の親節点
$\psi(m)$:　メソッド m からメソッド AFGM への射影
$e =_{id} e'$:　式 e と式 e' が同じ識別子であるとき**真**，さもなくば**偽**

図 6　定　　義
Fig. 6　Definitions.

直接呼び出されるメソッドとは，Method 節点に対応するメソッドを指す(注2)．間接的に呼び出されるメソッドとは，直接呼び出されるメソッドが内部で呼び出す可能性のある同一インスタンスのメソッドを指し，MFG を用いて導出する．

またオブジェクトコンテクストは，FS オブジェクト

(注2)：メソッドのオーバライドが行われた場合，$\mathcal{P}$ の型が唯一に特定できなければ同一シグニチャに対してメソッドが複数存在する可能性がある．

```
public class Calc {
  Integer i;
  public Calc() {i = new Integer(0);}
  public void inc() {i=new Integer(i.intValue()+1);}
  public void add(int c) {
    i = new Integer(i.intValue() + c);
  }
  public Integer result() {return(i);}
}
```

(a) FI オブジェクトコンテクスト

```
public class Calc {
  Integer i;
  public Calc() {i = new Integer(0);}
  public void inc() {i=new Integer(i.intValue()+1);}
  public void add(int c) {
    i = new Integer(i.intValue() + c);
  }
  public Integer result() {return(i);}
}
```

(b) FS オブジェクトコンテクスト

図 7　オブジェクトコンテクスト
Fig. 7　Object context.

コンテクスト（Flow Sensitive Object Context），FI オブジェクトコンテクスト（Flow Insensitive Object Context）に分けられる．前者はメソッドの実行順を考慮するが，後者は考慮しない．そのため，FS オブジェクトコンテクストは FI オブジェクトコンテクストより正確なエイリアス情報を抽出することができる．例として，図 7 において Calc::Calc()，Calc::add()，Calc::result() が順に呼ばれたときの**return**(i) に関するエイリアス計算を挙げる．図 7(b) はメソッド実行順を考慮しているため，図 7(a) と比較してそのエイリアス集合は小さく，Calc::Calc() メソッド内の 2 式 i，**new** Integer が除外されているのがわかる．メソッド実行順を考慮するには解析コストの増加は避けられず，これらはトレードオフの関係となる．なお，本論文では FI オブジェクトコンテクストを採用している．

3.5　エイリアス計算例

エイリアス計算の例として，図 8 の参照変数 c（太枠網掛部）のエイリアスの抽出手順を説明する．

（1）　参照変数 c に対応する AFG 節点から AFG 辺をたどり，Method 節点 result() に到達（図 9）

（2）　Method 節点 result() は親節点 b をもつため，親節点 b のエイリアス計算を行い，Method 節点 result() のエイリアス計算に関与するインスタンスを特定する

```
public class Calc {
  Integer i;
  public Calc() {
    i = new Integer(0);
  }
  public void inc() {
    i = new Integer
      (i.intValue() + 1);
  }
  public void add(int c) {
    i = new Integer
      (i.intValue() + c);
  }
  public Integer result() {
    return(i);
  }
}
```

```
class Test {
  Calc a, b;
  Integer c;
  Test() {
    a = new Calc();
    b = new Calc();
    a.inc();
    b.add(1);
    c = b.result();
  }
}
```

図 8　文 c = b.result() の c に関するエイリアス（網掛部）
Fig. 8　Alias set for c at c = b.result()(Masked).

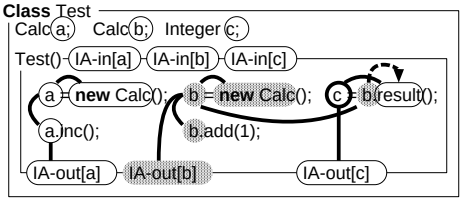


図 9　図 8 の文 c = b.result() の b に関するエイリアス（網掛部）
Fig. 9　Alias set for b at c = b.result() in Fig. 8 (Masked).

（a）　節点 b のエイリアス $\mathcal{B}$ を計算（図 9）

（b）　$\mathcal{B}$ に存在するインスタンス生成式から節点 b の型を特定 [Calc クラス]

（c）　Object-Context($\mathcal{B}$) を計算 [{Calc::Calc(), Calc::add(**int** c)，Calc::result()}]

（3）　節点 b は Calc クラスのインスタンスへの参照であることから，Calc::result() の MA-out 節点から AFG 辺をたどる（図 10）

（a）　AFG 辺をたどり，IA-in[i] に到達

（b）　属性 i のエイリアスに影響を与え得るメソッド，すなわち Object-Context(**this**) = Object-Context($\mathcal{B}$) の IA-out[i] 節点から AFG 辺をたどる（Calc::Calc()，Calc::add() も同様）

図 8 に，c のエイリアス（網掛部）及び Object-Context($\mathcal{B}$)（下線部）を示す．Object-Context($\mathcal{B}$) に含まれない Calc::inc() はエイリアス計算対象から除外されているのがわかる．

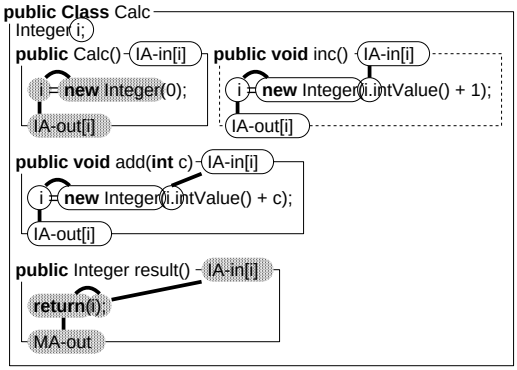


図 10　図 8 の文 c = b.result() の b.result() に関するエイリアス（網掛部）
Fig. 10　Alias set for b.result() at c = b.result() in Fig. 8 (Masked).

4. 実　　験

提案手法をJava を対象言語とするエイリアス解析ツールとして実装し，本手法の有効性を評価した．解析アルゴリズムとして，FI オブジェクトコンテクストを採用している．

4.1 Java エイリアス解析ツール

ツールは解析部（Java エイリアス解析ライブラリ）とユーザインタフェース部（Java エイリアス表示ツール）で構成されており，その構成を図 11 (a) に示す．以降，解析部及びユーザインタフェース部をそれぞれ簡単に説明する．

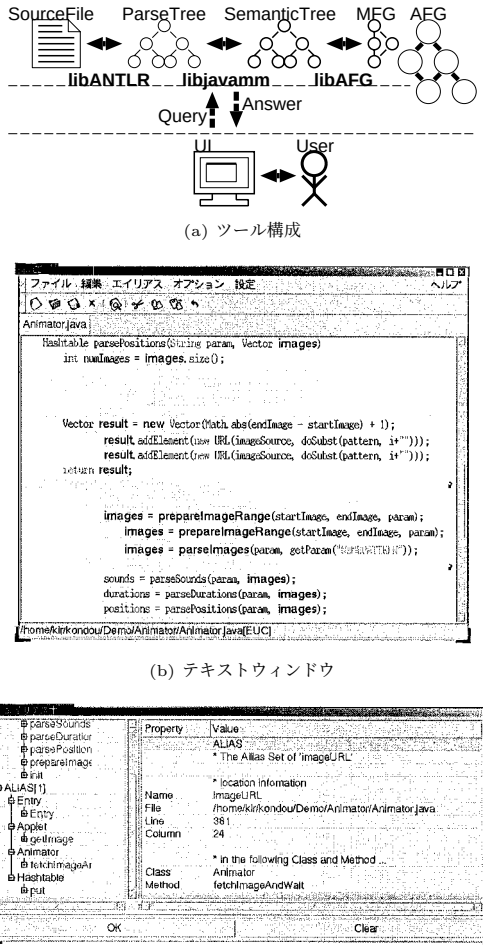
[解析部]

解析部は C++で記述されており，libANTLR（字句解析，構文解析）(注3)，libjavamm（意味解析），libAFG（エイリアス解析）の三つのライブラリで構成されている．これらのライブラリにより，各Java ソースプログラムに対して構文解析木，意味解析木，MFG 及び AFG がそれぞれ構築される．

[ユーザインタフェース部]

ユーザインタフェース部は，C++で記述されており Gtk−− [6] ツールキットを使用している．その機能には，テキストウィンドウ（図 11 (b)）及びエイリアストリーウィンドウ（図 11 (c)）によるエイリアス表示，プログラム編集がある．ユーザがあるオブジェクトへの参照式に関するエイリアスの抽出を指示すると，ツールはテキストウィンドウ及びエイリアストリーウィンドウ上にその解析結果を表示する．

エイリアス集合の導出により，プログラムテキストはエイリアス集合に含まれるエイリアス部分（Alias part）と，エイリアス集合に含まれない非エイリアス部分（Non-alias part）に区分される．テキストウィンドウでは，ユーザの視点をエイリアス部分に着目させなければならないが，エイリアス部分と非エイリアス部分の差別化の手段はユーザの目的によって様々である．とりわけ，エイリアスの性質上ソースコード中に占める割合は非エイリアス部分が圧倒的に多く，その視覚化に重点をおく必要がある．本ツールでは目的に応じた三つの視覚化手法を実現し，その一つとして，図 11 (b) ではエイリアス部分を背景色の変更により，非エイリアス部分を線分化により差別化を行っている．



(a) ツール構成

(b) テキストウィンドウ

(c) エイリアストリーウィンドウ

図 11　Java エイリアス解析ツール
Fig. 11　Java alias analysis tool.

(注3)：ANTLR [5] は，言語 L の文法 $\mathcal{L}$ を与えることで L の字句解析，構文解析ルーチンを C++若しくはJava で生成するツールである．

表 2　対象プログラム
Table 2　Target programs.

プログラム	新規		再利用可能（JDK）	
[概要]	ファイル	行	ファイル	行
WeirdX	47	16,703	815	115,977
[X サーバ]	(5.5%)	(12.6%)	(94.5%)	(87.4%)
ANTLR	129	18,775	267	33,847
[構文解析生成]	(32.6%)	(35.7%)	(67.4%)	(64.3%)
DynamicJava	242	32,037	825	119,564
[インタプリタ]	(22.7%)	(21.1%)	(77.3%)	(78.9%)

表 3　エイリアス解析時間（Phase 1〜3）[ms]
Table 3　Alias analysis time (Phase 1 – 3)[ms].

プログラム	新規	再利用可能
WeirdX	14,220	100,540
ANTLR	12,830	23,480
DynamicJava	56,260	110,150

PentiumIII-667 MHz-512 MB(Linux)

表 4　エイリアス計算時間（Phase 4）[ms]
Table 4　Alias calculation time (Phase 4)[ms].

プログラム	計算対象クラス	計算時間
WeirdX	com.jcraft.weirdx.Client	0.29
ANTLR	antlr.MakeGrammar	0.17
DynamicJava	koala.dynamicjava.interpreter.TypeChecker	0.07

また，エイリアス部分は複数メソッドにわたって点在することが多く，複数のファイルに及ぶことも少なくない．エイリアストリーウィンドウは，テキストウィンドウでは困難なエイリアス部分全体の把握を支援する．エイリアストリーの各節点は，クラス名，メソッド名，オブジェクトへの参照式を表す．図 11 (c) では，二つのエイリアス集合（ALIAS[0]，ALIAS[1]）のエイリアストリーが示されており，トリーの各節点を選択することで対応する要素の名前，型，位置などの情報を得ることができる．

4.2 計測方法

対象プログラムの概要を表 2 に示す．ここでは，対象プログラム自身を新規モジュール，対象プログラムが利用する JDK 1.2 付属クラスライブラリを再利用可能モジュールとみなし，以下の 3 点について検証を行った．

（1） 解析結果のモジュール化による効率化

新規モジュール及び再利用可能モジュールのエイリアス解析（Phase 1〜3）時間の比較（表 3）．

（2） AFG によるエイリアス計算時間

対象ファイルに存在するすべての AFG 標準節点からの平均エイリアス計算（Phase 4）時間（表 4）．

（3） 同一クラスの異なるインスタンスの属性を区別することによる解析精度への影響

表 5　エイリアス集合の要素数
Table 5　Size of alias set.

プログラム	計算対象クラス	非共有	共有
WeirdX	com.jcraft.weirdx.Client	15.37	24.54
ANTLR	antlr.MakeGrammar	5.94	18.77
DynamicJava	koala.dynamicjava.interpreter.TypeChecker	9.16	17.19

インスタンス属性に関するエイリアス情報に関し，

- インスタンスごとに区別しない（共有）
- インスタンスごとに区別する（非共有）

の二つの手法を用いて（2）と同様のエイリアス計算を行い，導出されたエイリアス集合の平均要素数を比較（表 5）．

5. 考　　察

5.1 実験データの解釈

解析結果のモジュール化による効率化（表 3）に関して，到達エイリアス集合のみによるエイリアス解析では，モジュール単位でエイリアス解析結果を保持することができないため，あるモジュールが変更されるとそのモジュールが利用する他のモジュールも再解析の対象にせざるを得ない．本手法では，新たな変更が加えられず繰り返し再利用されるモジュールを解析し AFG を構築しておくことで，ユーザは新規・更新モジュールのみ解析すればよく，解析コストの削減が得られる．

到達エイリアス集合のみによるエイリアス解析では，解析終了時点ですべてのエイリアス集合が抽出されている．提案手法のエイリアス解析過程は，AFG 構築（Phase 1〜3）及び AFG によるエイリアス計算（Phase 4）に区分されており，エイリアス集合を導出するにはエイリアス計算を行う必要がある．しかし，AFG 構築時間と比較した場合，エイリアス計算に要する時間コストは小さく，求めるエイリアスが特定のものに限られている場合やユーザとの対話形式でエイリアスを求める解析ツールの実現においては，実用に十分に足るものであると考えている（表 4）．

また本手法では，同一クラスのインスタンスに共通するエイリアス関係のみ AFG に保持し，インスタンス独自のエイリアス関係はエイリアス計算時に導出する．本手法の適応により，同一クラスの異なるインスタンスのエイリアス情報を個別に取り扱うことで精度の向上が得られている（表 5）．

```
   void main() {
1:   char *a, *b;
2:   char c = ' ';
3:   a = &c;
4:   assign(&b, a);
5:   putc(*b);
   }

6: void assign(char **y,
         char *x) {
7:   *y = x;
   }
```

```
((3, &c), (3, a))
((3, a), (4, a))
((1, b), (4, b))
((4, &b), (4, AA-in[y]))
((4, b), (4, AA-in[*y]))
((4, a), (4, AA-in[x]))
((4, AA-out[*y]), (4, b))
((4, b), (5, b))

((6, FA-in[y]), (7, y))
((6, FA-in[*y]), (7, *y))
((6, FA-in[x]), (7, x))
((7, x), (7, *y))
((7, *y), (6, FA-out[*y]))
```

図 12　ポインタ変数をもつ言語（C）でのエイリアス解析
Fig. 12　Alias analysis for languages with pointer variables like C.

5.2　ポインタ変数をもつ言語でのエイリアス解析

3. で述べたアルゴリズムはJAVA のもつ参照型の式によるエイリアスを仮定したものであったが，C, C++などポインタ変数をもつ言語ではポインタ（特に高階ポインタ）による間接参照を考慮する必要がある．これは，引き数として n 階 ($n \geq 2$) ポインタ変数を使用することで，たとえ値渡しであっても，手続き内で呼出し元のエイリアス関係を変更可能であることによる．

提案手法をポインタに適用するため Phase 1，2，4 を拡張する．以下にその拡張に関して簡潔に述べる．

— Phase 1：AFG 節点の抽出，Phase 2: AFG 辺の抽出

参照変数では 1 変数当り一つのエイリアス情報であったが，n 階ポインタ変数では 1 変数当り n 個のエイリアス情報を用意する．図 12 は，C 言語で記述されたプログラム及びそこから抽出された直接のエイリアス関係の集合を示している．手続き assign(**char** **y, **char** *x) は 2 階のポインタ変数 y を使用しており，実引き数である&b について&b 及び b に関するエイリアス情報が AA-in 節点として，仮引き数 y について y 及び *y に関するエイリアス情報が FA-in 節点として，手続き main()，手続き assign(**char** **y, **char** *x) にそれぞれ用意されている．また，b, *y に関するエイリアス情報を呼出し先に反映するため，AA-out，FA-out 節点がそれぞれ追加されている．

— Phase 4：エイリアス計算

ポインタに関する演算子としては，間接演算子 '$*$' 及びアドレス演算子 '&' がある．$*x$ は x が指す記憶領域の内容を，$\&x$ は x の記憶領域の位置を表す．エイリアス計算アルゴリズムをこれらの演算子に対応させるため，図 5 の**Step 2** を拡張する．詳細は，図 13 に示す．

Step 2: E を始点とし，新たな到達可能節点が見つからなくなるまで AFG 辺を推移的にたどる
加えて，到達節点 C の種類に応じて以下のような処理も行う
[Cond.1] 親節点をもつ標準（インスタンス属性）節点：
……
[Cond.8] '$*$' 演算子をもつ AFG 標準節点 ($*x = \varphi^{-1}(C)$)：
（1） $X := \varphi(x)$
（2） $\{\varphi(y) \mid \varphi(\&y) \in \mathcal{X}\}$ から AFG 辺をたどる
$\{\varphi(*y) \mid \varphi(y) \in \mathcal{X}\}$ から AFG 辺をたどる
[Cond.9] '&' 演算子をもつ AFG 標準節点 ($\&x = \varphi^{-1}(C)$)：
（1） $X := \varphi(x)$
（2） $\{\varphi(y) \mid \varphi(*y) \in \mathcal{X}\}$ から AFG 辺をたどる
$\{\varphi(\&y) \mid \varphi(y) \in \mathcal{X}\}$ から AFG 辺をたどる

図 13　エイリアス計算アルゴリズム（ポインタ）
Fig. 13　Alias computation algorithm (for pointers).

5.3　関 連 研 究

エイリアス解析のモジュール化に関する研究としては [10], [15] がある．前もってモジュール内のエイリアス解析を行う点で我々と同じであるがその計算の手間は大きくなる．これらの手法では各文の到達エイリアス集合の要素をエイリアス関係の成り立つ組 $\{(\alpha, \beta), (\gamma, \delta)\}$ で表現している．これは，α，β がエイリアス関係を満たすならば，(γ, δ) がエイリアスとして導出されることを意味する．このとき，(α, β) を (γ, δ) に対する制約条件という．この制約条件は，本論文でいう間接的なエイリアス関係の成立条件に対応する．

しかし，これらの手法はエイリアス解析のモジュール化（呼出し側モジュールの解析結果に依存しないエイリアス解析）を実現するため，各モジュール内でのエイリアス解析時に，対象モジュールから参照可能な外部変数（大域変数や仮引き数など）のすべての組合せを (α, β) として与え，そのとき成立するエイリアス組 (γ, δ) を抽出する必要がある．

我々の手法は直接的なエイリアス関係のみを AFG 辺で表現し，間接的なエイリアス関係は AFG 上での到達可能性を調べることで得るようにしている．したがって，モジュール内のエイリアス解析時（Phase 1〜2）には，外部変数すべての組合せを想定する必要はなく，直接的なエイリアス関係を満たす節点間に AFG 辺を引くだけでよい．モジュールにまたがる解

析（Phase 4）のとき，外部変数間のエイリアス関係は到達可能な AFG として与えられる．モジュール内解析ではエイリアスとして導出されなかった二つの式が（外部からの影響により）間接的なエイリアス関係を満たすかどうかは，与えられた外部からの AFG を経由することで到達可能となるかを調べればよい．

また，[10] は手続き型言語に関してのみ論じられており，オブジェクト指向言語特有の問題，例えば本論文で扱うオブジェクトコンテクストなどは考慮されていない．[15] は参照変数への代入により発生するエイリアスを求める方法を提案しており，ポインタ変数によるエイリアスには触れられていない．また，いずれの研究もプロトタイプ実装を行っただけで，実用的なシステム構築には至っていない．

一方，今回我々は FI オブジェクトコンテクストを採用しており，メソッドの呼出し順を考慮している上記 2 手法と比較して解析精度が低下する場合がある．精度とコストはトレードオフの関係にあるため一概に手法の優越を判断することは難しいが，今後 FS オブジェクトコンテクストも加えた相互比較を行いたいと考えている．

6. むすび

本研究では，オブジェクト指向言語に対する，再利用性，モジュール性を考慮したエイリアスフローグラフによるエイリアス解析手法の提案を行った．また，提案手法を Java エイリアス解析ツールとして実装を行い，その有効性を検証した．今後の課題としては，既存のエイリアス解析手法とのコストや精度に関する比較，AFG データベースの構築，例外処理，スレッドへの対応などが挙げられる．

謝辞 本研究は，栢森情報科学振興財団の補助を受けている．

文　献

[1] A.V. Aho, R. Sethi, and J.D. Ullman, Compilers: Principles, Techniques, and Tools, Addison-Wesley, 1986.
[2] B. Steensgaard, “Points-to analysis in almost linear time,” 23rd Annual ACM SIGACT-SIGPLAN Symposium on the Principles of Programming Languages, pp.32–41, 1996.
[3] B. Stroustrup, The C++ Programming Language, Third edition, Addison-Wesley, 1997.
[4] G. Booch, Object-Oriented Design with Application, The Benjamin/Cummings Publishing Company, 1991.
[5] http://www.ANTLR.org/, ANTLR Website.
[6] http://gtkmm.sourceforge.net/, Gtk−−.
[7] J. Gosling, B. Joy, and G. Steele, 村上雅章（訳），The Java 言語仕様，1997.
[8] 片山卓也，土居範久，鳥居宏次（監訳），ソフトウェア工学大事典，朝倉書店，1998.
[9] M. Enami, R. Ghiya, and L.J. Hendren, “Context-sensitive interprocedural points-to analysis in the presence of function pointers,” SIGPLAN’94 Conference on Programming Language Design and Implementation, pp.242–256, 1994.
[10] M.J. Harrold and G. Rothermel, “Separate computation of alias information for reuse,” IEEE Trans. Software Eng., Special Section of Best Papers of the 1996 International Symposium on Software Testing and Analysis, vol.22, no.7, pp.442–460, 1996.
[11] M. Hind and A. Pioli, “An empirical comparison of interprocedural pointer alias analysis,” IBM Research Report #21058, 1997.
[12] M. Shapiro and S. Horwitz, “Fast and accurate flow-insensitive point-to analysis,” 24th Annual ACM SIGACT-SIGPLAN Symposium on the Principles of Programming Languages, 1997.
[13] M. Weiser, “Program slicing,” Proc. 5th International Conference on Software Engineering, pp.439–449, 1981.
[14] P. Tonella, G. Antoniol, R. Fiutem, and E. Merlo, “Flow insensitive C++ pointers and polymorphism analysis and its application to slicing,” Proc. 19th International Conference on Software Engineering, pp.433–443, 1997.
[15] R.K. Chatterjee and B.G. Ryder, “Modular concrete type-inference for statically typed object-oriented programming languages,” Department of Computer Science, no.DCS-TR-349, Rutgers University, 1997.
[16] R.P. Wilson and M.S. Lam, “Efficient context-sensitive pointer analysis for C programs,” SIGPLAN’95 Conference on Programming Language Design and Implementation, pp.1–12, 1995.

（平成 12 年 7 月 7 日受付，11 月 6 日再受付）

大畑　文明

平 10 阪大・基礎工・情報中退．平 12 同大大学院修士課程了．現在，同大学院博士課程在学中．プログラム構造解析の研究に従事．

近藤　和弘

平 12 阪大・基礎工・情報卒．現在，同大大学院修士課程在学中．プログラム構造解析の研究に従事．

井上　克郎　（正員）

昭 54 阪大・基礎工・情報卒．昭 59 同大大学院博士課程了．同年同大・基礎工・情報助手．昭 59〜61 ハワイ大マノア校・情報工学科助教授．平 1 阪大・基礎工・情報講師．平 3 同学科助教授．平 7 同学科教授．工博．ソフトウェア工学の研究に従事．