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.
SS2002-56 (2003-03)

エイリアス関係を利用したJavaスライシングツールの実現

山中　祐介†　　横森　励士†　　井上　克郎††

† 大阪大学大学院基礎工学研究科　〒560-8531 大阪府豊中市待兼山町 1-3
†† 大阪大学大学院情報科学研究科　〒560-8531 大阪府豊中市待兼山町 1-3
E-mail: †{y-yamank,yokomori}@ics.es.osaka-u.ac.jp, ††inoue@ist.osaka-u.ac.jp

あらまし　近年の大規模化・複雑化するプログラムに対して，プログラムスライスと呼ばれる手法がプログラムの理解，デバッグの効率化を目的として提案されている．プログラムスライスとは，プログラム文中においてある文のある変数の値に影響を与える文の集合を抽出する技術で，プログラム中の特定の機能やエラーに関係がある部分を抽出するのに利用される．プログラムスライスは，一般にプログラム文間の依存関係解析を行うことで得られるが，従来のオブジェクト指向言語における手法では同一メモリ空間を指す可能性のある式間の同値関係であるエイリアス関係を十分に考慮されておらず，エイリアス関係の解析について曖昧なものがほとんどである．本論文では，オブジェクト指向言語 Java を対象として，エイリアス関係を考慮した静的プログラムスライス計算手法を提案する．エイリアス関係を利用することで，オブジェクト指向言語特有の実行時決定要素の解析を含めた，より正確なプログラムスライスの計算が期待できる．また，提案手法を我々の研究グループが開発している Java プログラム解析フレームワークにスライシングツールとして機能追加し，その有効性を解析コスト及び正確性の面から確認する．
キーワード　プログラムスライス，エイリアス，静的解析，Java

Java Slicing Tool Based on Alias Analysis

Yuusuke YAMANAKA†, Reishi YOKOMORI†, and Katsuro INOUE††

† Graduate School of Engineering Science, Osaka University
1-3, Machikaneyama-cho, Toyonaka-shi, Osaka 560-8531, Japan
†† Graduate School of Information Science and Technology, Osaka University
1-3, Machikaneyama-cho, Toyonaka-shi, Osaka 560-8531, Japan
E-mail: †{y-yamank,yokomori}@ics.es.osaka-u.ac.jp, ††inoue@ist.osaka-u.ac.jp

Abstract　Program slicing is a technique that extracts program statements which affect the value of variable in certain statement, and is very promising approach for program debugging, testing, understanding, merging, and so on. In order to calculate a program slice, we must know the dependence relations between statements in the program. When an expression refers to a memory location that is referred to by another expression, we say that there is an alias relation between those expressions. Although many researchers have already proposed slice-calculation methods and implemented prototype tools for object-oriented programs, consideration for the alias relation is insufficient. In this paper, we propose Java slicing system based on alias analysis. In OO-language such as Java, there are a lot of elements which are determined dynamically, so caluculation of program slice becomes more exact by using the alias relation. Moreover, we implement a slicing system which performs in Java program analysis framework, and evaluate the proposal method.
Key words　Program Slice, Alias, Static Analysis, Java

1. まえがき

プログラミング言語の高級化，ソフトウェアの大規模化・複雑化に伴い，プログラムの理解やデバッグがより困難なものになりつつある．そのため様々なプログラム解析手法の提案及び実装がなされており，その一手法としてプログラムスライシングがある．

プログラムスライシングは Weiser [7] により提案された手法

であり，フログラム中のある文 s における変数 v に対して v の値に影響を与え得る全ての文を抽出する技術で，抽出された文の集合をフログラムスライス，または単にスライスと呼ぶ．スライスを用いることで，フログラムに存在する特定の機能やエラーに関係がある部分を抽出することができ，その結果，フログラムの理解やデバッグにかかる負担の軽減でき，開発効率の向上が期待できる．

一般に，スライスはプログラム文間の制御構造及び変数の参照・定義情報から得られる依存関係によって構築されるフログラム依存グラフ [11] を用いて計算される．依存関係を解析する際に，エイリアス解析などの手法を組み入れることでプログラム依存グラフをより正確に構築することができる．エイリアス関係とはフログラム上の式の対が同一メモリ領域を指すことであり，その式の集合をエイリアス集合と呼ぶ．エイリアス集合を静的に求めることをエイリアス解析という．エイリアス関係は，引数の参照渡し，参照変数，ポインタを介した間接参照などで生じるため，参照変数が数多く存在するオブジェクト指向言語の Java などにおいては，フログラム理解に効果的に利用することができる．

オブジェクト指向言語などの，エイリアス関係が存在しうるフログラムに対してスライス計算を行うためには，依存関係解析において変数の参照・定義の情報を用いる際にその変数がどのエイリアス集合に属しているかを知ることが必要不可欠である．これまでオブジェクト指向言語に対して様々なスライス計算手法の提案及び実装がなされてきたが，既存の手法はフログラム中に存在するエイリアス関係を十分に考慮されておらず，エイリアス関係が考慮されていてもその実装について曖昧にしているものがほとんどである．

そこで，本論文ではオブジェクト指向言語において，エイリアス関係を考慮した静的スライス計算手法を提案する．オブジェクト指向言語には，数多くのフログラム実行時に決定される要素があり，静的にフログラムの解析を行うためにはエイリアス解析が有効 [2] [8] とされている．そのため，オブジェクト指向言語を対象としたスライス計算には，エイリアス関係を考慮することが重要であると考えられる．エイリアス関係を利用することで，既存の手法では抽出することができなかった依存関係の抽出が可能になる．その結果，より精度の高いフログラムスライスの計算を行うことが期待できる．

また，提案手法の実装を Java に対して行う．実装においては，我々の研究グループで開発を行っている Java フログラム解析フレームワークにスライシングツールを機能拡張することで実現し，その有効性を得られた解析コスト，スライスサイズの 2 点から確認する．

以降，2. ではフログラムスライス及び既存の計算手法について述べ，3. ではエイリアスについて述べる．4. では提案手法について述べ，5. で Java フログラム解析フレームワークへの提案手法の実装について述べる．6. で提案手法と実装したスライシングツールの評価について述べ，最後に 7. でまとめと今後の課題について述べる．

2. プログラムスライス

プログラムスライシング（*Program Slicing*）とは，**スライス基準**（*Slice Criterion*，対 $< s, v >$ で示され，s は文，v は s で定義もしくは参照される変数を表す）に影響を与え得る全ての文をフログラムから抽出する技術で，その結果取り出された文の集合を**プログラムスライス**（*Program Slice*）または単に**スライス**（*Slice*）と呼ぶ．Weiser [7] により提案されたフログラムスライスは，フログラム中に存在するフォールトの位置特定に有効であるだけでなく，フログラム保守や理解などにも利用できることが知られている [6] [7] [10]．

2.1 プログラムスライス計算法

スライス計算にはさまざまな手法が提案されてきたが，本論文ではフログラム依存グラフによるスライス抽出技法 [11] に着目する．この手法は，次の 4 フェーズで構成されており，以下各フェーズについて簡単に述べる．

□ Phase1: 定義，参照変数の抽出

フログラムの各文で定義，参照されている変数を抽出する．

□ Phase2: 依存関係解析

Phase1 の結果を元に，フログラム文間に存在する**制御依存関係**（*Control Dependence Relation, CD Relation*），**データ依存関係**（*Data Dependece Relation, DD Relation*）[11] という 2 つの依存関係を抽出する．

- **制御依存関係:** フログラム中の 2 文 s, t に関して，以下の条件を満たすときに存在する．

（1） s は条件文（節）

（2） t の実行は s の判定結果に依存する

- **データ依存関係:** フログラム中の文 s から文 t の間に変数 v に関して，以下の条件を満たすときに存在する．

（1） s は v を定義する

（2） t は v を参照する

（3） s から t への 1 つ以上の実行経路の中に，v の再定義が起こらない経路が少なくとも 1 つ存在する

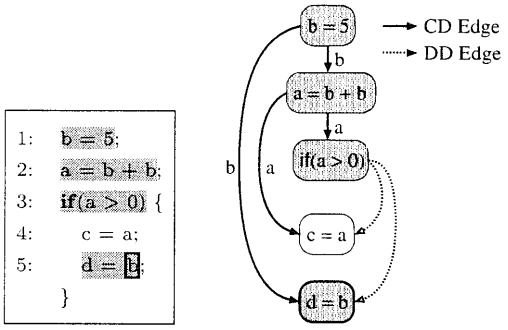
□ Phase3: プログラム依存グラフの構築

Phase2 で抽出された依存関係を利用し，フログラム依存グラフを構築する．**プログラム依存グラフ**（*Program Dependence Graph, PDG*）[11] とは，フログラム内の文間の依存関係を表す有向グラフであり，その節点は，フログラムに存在する文（条件判定文，代入文，入出力文，手続き呼出文）を表し，その有向辺は 2 つの節点間の制御依存関係，データ依存関係（それぞれ**制御依存辺**（*Control Dependence Edge, CD Edge*），**データ依存辺**（*Data Dependence Edge, DD Edge*）と呼ぶ）を表す．

□ Phase4: プログラム依存グラフによるスライス抽出

スライス基準に対するスライスを抽出する．スライス基準 $< s, v >$ に対するスライスとは，s に対応した PDG 中の節点 N_s から，逆方向に制御依存辺及びデータ依存辺を経て推移的に到達可能な節点集合に対応する文の集合をいう．

図 1（a）にサンフルフログラムとスライス基準 <5, b>（太枠部）に対するスライス（網掛部）を，そして図 1（b）に対応する PDG を示す．



(a) サンプルプログラム　　(b) PDG

図 1　スライスの例

2.2　スライスの静的解析と動的解析

スライスには，依存関係解析のフェーズにおける解析手法の違いにより，**静的スライス**（*Static Slice*）[7] と**動的スライス**（*Dynamic Slice*）[3] の 2 種類に大別される．

□ **静的スライス**

静的スライスとは，**静的解析**（*Static Analysis*）に基づき，入力に関しては全ての可能性を考慮し，実行制御においては全ての分岐に実行が遷移しうるとして文間の依存関係が抽出される．その特徴としては，入力データがソースコードのみで十分であること，節点が文単位でよいので動的スライスに比べてはるかにコストが小さい．しかし，プログラム実行においてすべての実行経路が利用されるとは限らないため，実行時エラーの原因を把握するためのフォールト位置特定に対しては必ずしも効果的とはいえない．本論文ではこの静的スライスに着目する．

□ **動的スライス**

動的スライスとは**動的解析**（*Dynamic Analysis*）に基づき，ある特定の入力データによる実行系列間の依存関係が抽出される．その特徴としては，解析対象を特定の実行経路に限定しているため，静的スライスに比べてスライスのサイズが小さくなるというところにある．また，実際に実行された部分の中からのみスライスが計算されるため，フォールト位置特定を効率よく行なうことができる．しかし，動的スライスの計算には，節点が実行系列であるので，実行系列と依存関係を全て記憶しなければならないため，多大な時間及び空間コストを要するという問題点がある．

3.　エイリアス

エイリアス（*Alias*）とは，引数の参照渡し，参照変数，ポインタを介した間接参照などで生じる，同じメモリ領域（オブジェクト）を指す可能性のある式間の同値関係であり，**エイリアス関係**（*Alias Relation*）ともいう．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，プログラムの静的解析を行うことで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，コンパイラ最適化 [1] や，プログラムスライスの計算に欠くことのできないものである．図 2 にエイリアスの例を示す．網掛部と太枠部のそれぞれが独立したエイリアス集合である．例えば，5 行目と 7 行目の変数 c は別のエイリアス集合に属しているので，それぞれ別のメモリ領域を指している．

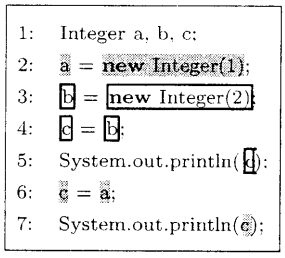


図 2　エイリアスの例

3.1　エイリアスとプログラムスライス

変数アドレスの参照が存在するプログラムにおいては，エイリアスの存在により，プログラムの異なるスコープ中の異なる識別子が同じメモリ領域を指す可能性がある．そのため，静的スライスにおける計算では **Phase2:データ依存関係解析**において，どの変数が定義・参照されているかだけでなくその変数がエイリアス集合に含まれているかという情報が必要になる．エイリアス関係を無視して依存関係を抽出すると，エイリアス関係により異なる変数で同一のメモリ空間を変更した場合などにおいて，データ依存関係を正しく把握できないため，著しく正確性に欠く場合がある．

3.2　オブジェクト指向言語におけるエイリアス

オブジェクト指向言語（*Object Oriented Language*）には，従来の手続き型言語にはない**クラス**（*Class*），**オブジェクト**（*Object*），**継承**（*Inheritance*），**動的束縛**（*Dynamic Binding*），**多態**（*Polymorphism*）などの概念が導入されている．これら概念はプログラムの実行時に決定される要素であり，オブジェクト指向言語を静的にプログラムの解析を行うためにはエイリアス解析が有効 [2] [8] で不可欠なものであるとされている．

3.3　FI エイリアス解析と FS エイリアス解析

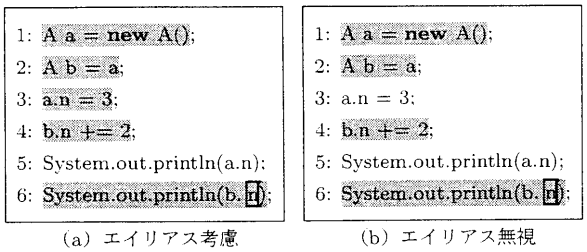
エイリアス集合を導き出すエイリアス解析は，大きく **FI エイリアス解析**（*Flow-Insensitive Alias Analysis*），**FS エイリアス解析**（*Flow-Sensitive Alias Analysis*）の 2 つに分けることができる．以下，それぞれについて説明する．

□ **FI エイリアス解析**

FI エイリアス解析 [2] とは，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**（*Alias Graph*）を利用する．エイリアスグラフは無向グラフであり，グラフの節点はメモリ領域を指す可能性のある変数及び式を，辺は代入や引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．

□ **FS エイリアス解析**

FS エイリアス解析 [9] 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, RAset*）を利用する．プログラムのある文 s における到達エイリアス集合 $RA(s)$ の要素は，次の条件を満たすエイリアス集合である．



(a) エイリアス考慮　　(b) エイリアス無視

図 3　スライスの比較

（1） 文 s の実行直前に成立している

（2） 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る

到達エイリアス集合の各要素は (文番号, 式) の組で表される．本論文ではこの FS エイリアス解析に着目しており，図 2 の例は FS エイリアス解析によって得られた結果である．

4. エイリアス関係を考慮したプログラムスライス

3. 節で述べたとおり，参照が生じるフログラム言語のフログラムスライスの計算を行う際，エイリアス解析が必要不可欠である．エイリアス関係を考慮しなければならない例として，図 3 のサンフルフログラムにおけるスライス基準 <6, n>（太枠部）に対するスライス（網掛部）をエイリアスを考慮した場合は図 3 (a) に，エイリアスを無視した場合は図 3 (b) に示す．変数 a, b は 2 行目の演算によってエイリアスが発生してるので，3, 4 行目で定義・参照されているメンバ変数 n は同じメモリ領域にある．そのため，スライス基準であるメンバ変数 n は変数 a だけでなく変数 b にも影響を受けている．よって，3 行目もスライスに含まれるべきである．このようにエイリアス関係を無視した場合，得られるスライスが実際には間違っていることがある．

これまで様々なオブジェクト指向言語を対象としたスライス計算手法が提案されているが，エイリアス関係を十分に考慮されていない，もしくは考慮されていない．そこで，本論文ではエイリアス関係を考慮した静的フログラムスライス計算手法を提案する．この手法における前提として，解析対象フログラムのエイリアス解析が終了しており，ある式のエイリアス集合の情報が取得可能になっているとする．エイリアス関係の情報を用いることによって，既存の手法では抽出することができないデータ依存関係の抽出を行うことが可能になる．

4.1 方　　針

□ Phase1, 2 の変更

エイリアス関係を考慮するためスライス計算の 4 つのフェーズのうち Phase1, 2 を変更する．

- **Phase1:** 変数の参照・定義の抽出に加え，フログラム文中の式がどのエイリアス集合に属しているかを抽出する．
- **Phase2:** エイリアス関係を考慮したデータ依存関係を抽出するために次に挙げる 3 つの表と，一つ一つのエイリアス集合を区別するための各エイリアス集合に対する ID を用意する．

ある文 s において，
入力： 定義されている変数 v
　　v がメンバ変数として属する型の ID i
出力： なし
if i が入力されている
　– $MT(i)$ から v に対応した $VT(v)$ を得る
– $VT(v)$ の文番号を s の文番号に変更
if v は参照変数である
　i を $VT(v)$ の ID の値に変更
　if エイリアス表 $AT(v)$ の ID と $VT(v)$ が不一致
　　– $VT(v)$ の ID の値を $AT(v)$ の ID の値に変更
　if メンバ変数表 $MT(i)$ が存在しない
　　– $MT(i)$ を作成，変数表の文番号は s の文番号

図 4　変数定義時のアルゴリズム

ある文 s において，
入力： 参照されている変数 v
　　v がメンバ変数として属する型の ID i
出力： v の ID i'
if i が入力されている
　– $MT(i)$ から v に対応した $VT(v)$ を得る
– $VT(v)$ の文番号に対応した文 t を得る，
　t から s へ v に関するデータ依存関係が存在することを抽出
if v は参照変数である
　i' を $VT(v)$ の ID の値にする
　if $AT(v)$ の ID と $VT(v)$ の ID が不一致
　　– $VT(v)$ の ID の値を $AT(v)$ の ID の値に変更，
　　i の値を $AT(v)$ の ID の値に変更
　if $MT(i')$ が存在しない
　　– $MT(i')$ を作成，変数表の文番号は s の文番号

図 5　変数参照時のアルゴリズム

– **変数表** $VT(v)$**:** 変数 v が最後に定義された文番号と，どのエイリアス集合に属しているかを表現する ID を持つ．

– **エイリアス表** $AT(e)$**:** 式 e がどのエイリアス集合に属しているかを表現する ID を持つ．

– **メンバ変数表** $MT(i)$**:** エイリアスの ID i に対応するエイリアス集合の型に属するメンバ変数それぞれに対応する変数表を持つ．

4.2 アルゴリズム

方針で挙げた 3 つの表（変数表，エイリアス表，メンバ変数表）を用いてデータ依存関係の抽出を行う．データ依存関係の計算において，処理を行わなければならないときはなんらかの変数が参照及び定義されているときであり，直接的なものとして各フログラム文での変数の参照及び定義，間接的なものとしてメソッド呼び出しによる対象メソッド内での参照，定義が挙げられる．そのため，変数の参照及び定義，そしてメソッド呼び出しの 3 つに関しての計算アルゴリズムが必要といえる．

ここでは，変数の定義におけるアルゴリズムとして図 4 を，変数の参照におけるアルゴリズムとして図 5 を示す．

ID i への入力があるのは変数 v が演算子"." の右辺である場合であり，このとき入力される i の値はその演算子"." の左辺の処理により出力結果である．例えば，$a.b$ とあれば b の処理を行うときに a の処理で出力される ID i' が入力される．

5. スライシングツールの実現

我々は，提案手法を我々の研究グループで開発している Java フログラム解析フレームワークに機能追加を行うことでスライシングツールを実現した．本節では，フレームワーク及び実現したスライシングツールの紹介を行う．

5.1 Java プログラム解析フレームワーク

Java フログラム解析フレームワークとは，我々の研究グループで開発を行っている静的に Java の解析を行うためのライブラリ，ツール群で，次に挙げる 4 つの部から構成されている．

□ **解析木構築部**

- **構文解析ライブラリ:** Java のソースコードを読み込み，**字句解析**（*Lexical Analysis*）及び**構文解析**（*Syntax Analysis*）を行い，抽象構文木を構築する．
- **意味解析ライブラリ:** 抽象構文木を読み込み，**意味解析**（*Semantic Analysis*）（識別子表を作成し，識別子に関する宣言と参照間の関係を抽出する）を行い，意味解析木を構築する．

□ **XML データベース部** [13]

Java フログラムの，構文木情報及び意味情報を **XML**（*eXtensible Markup Language*）を用いてデータベース化した部である．

- **XML-意味解析木 変換ライブラリ:** XML 文書と意味解析木の相互変換を行う．事前にソースコードを解析し XML 文書化していれば，ソースコードからではなく XML 文書からの解析が可能となる．
- **各種ツール:** XML–Java 変換，XML–HTML 変換，XML–XML 変換を行えるツールが提供されている．

□ **エイリアス解析部** [12]

意味解析木を読み込み，FS エイリアス解析を行う．また，ユーザの要求に応じたエイリアス集合の計算を行う．

- **エイリアス解析ライブラリ:** 意味解析木を読み込み，**エイリアスフローグラフ**（*Alias Flow Graph, AFG*）及び**メソッド呼び出しグラフ**（*Method Flow Graph, MFG*）を構築する．

エイリアスフローグラフとは，単一メソッド内のエイリアス関係を無向グラフで表現したものである．メソッド呼び出しグラフとは，クラス中に存在するメソッド間の呼び出し関係を有向グラフで表現したものである．

□ **ユーザインターフェース部**

ユーザインターフェースにはテキストの表示・編集機能及びエイリアス集合の計算結果のツリー表示機能がある．

5.2 スライシングツール

スライス計算部を Java フログラム解析フレームワークにライブラリとして追加実装することで，スライシングツールを実現した．スライシングツールを含めた Java フログラム解析フレームワークを図 6 に示す．

□ **スライス計算ライブラリ**

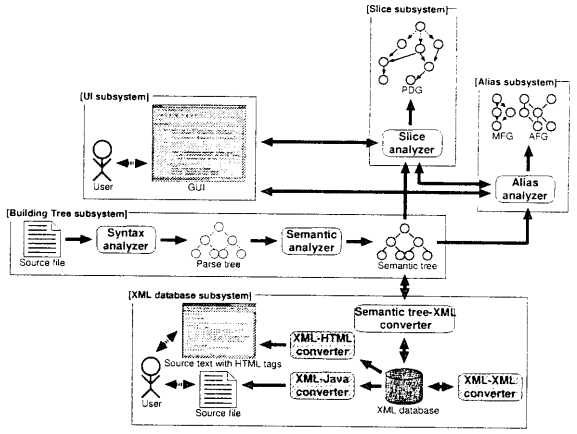


図 6 Java フログラム解析フレームワーク

意味解析木を読み込み，エイリアス解析部からエイリアス集合情報の抽出をしながらフログラム依存グラフを構築する．また，ユーザ要求として与えられたスライス基準に対するスライスの計算を行い，その結果をユーザインターフェース部に渡し表示する．

□ **オブジェクト指向言語への対応**

- **動的束縛，多態:** 実行時決定要素であるこれらの特性は，エイリアス解析により対象のオブジェクトやメソッドの型をある程度推測できる．
- **スレッド:** 一つの制御ブロックとして扱う．
- **例外:** 提案手法は単一メソッド内の解析を前提としており，例外のような複雑な制御構造は考慮していない．

□ **ツールの特徴**

- メソッド呼び出し文をスライス基準として指定することができる．これは，引数及びメソッド呼び出しが参照しているメンバ変数がスライス基準になっていると判断している．
- 抽出されるスライスの粒度を選択することが可能である．粒度の選択は以下の 2 種類から行える．
  - **詳細抽出:** スライス基準が参照変数である場合，メンバ変数を含めた参照変数全体のスライスを抽出
  - **簡略抽出:** スライス基準が参照変数である場合，メンバ変数を含めないスライス基準の参照変数そのものだけのスライスを抽出

6. 考　　察

6.1 評　　価

6.1.1 解析コスト

実装したスライシングツールを用いて実験を行った．実験の際には，CPU が Petium4 1.5Ghz，メモリ 512MB，OS が FreeBSD 5.0–CURRENT の PC を用いてソースコードの解析を行ってデータを取得した．ソースコード（1 ファイル）を解析するときの対象となるのは，そのソースコードが直接又は間接に参照しているソースコード全てであり，そこには JDK クラスライブラリも含まれる．ただし，PDG 構築に関しては JDK クラスライブラリを除いている．実験に使用したフログラムの

表 1　解析時間（単位：ms）

フログラム	意味解析木構築	エイリアス解析	PDG 構築
P1	12011.4	5175.8	18.8
P2	45611.4	21580.8	598.6
P3	12818.8	5302.8	1539.2

表 2　使用メモリ量（単位：MB）

フログラム	意味解析木	エイリアス	PDG
P1	132	16	1
P2	395	61	1
P3	135	14	7

概要は，実験用に作成したフログラム（P1，2 クラス，24 行），簡易ドローツール（P2，3 クラス，323 行），そして Apache が提供している XML ハーサ Xerces（P3，20 クラス，4302 行）である．解析時間，メモリ使用量の結果を表 1，表 2 に示す．

P2 は GUI を搭載しているため他と比べて JDK を含むソースコード量が大きい．その結果として，コストが非常に大きくなっている．Java フログラムの解析においては，解析対象が参照している JDK クラスライブラリの数が多くなりやすいため，その解析に多大なコストを要する場合が多い．そのため，JDK に対する PDG を構築を行えば解析コストが変わることが予想されるが，現時点での必要な解析時間は現実的なものだと考えられる．

6.1.2　スライスサイズ

P1 と P2 に対してエイリアス関係を考慮した場合としなかった場合とで，同じスライス基準に対してスライス計算を行った．計算結果として得られたスライスサイズの違いを表 3 に示す．

エイリアス関係を考慮してスライス計算を行った場合，エイリアス関係を無視する場合に比べてスライスサイズが P1, P2 共に大きくなっている．実際に抽出された部分の調査を行ったところ，参照変数の代入によりエイリアス関係が発生していた．そのため，その代入された変数に関する部分が新たにスライスとして抽出されており，エイリアスを無視していた場合のスライスは実際には正しくないものであった．このことから，エイリアス関係を利用することでフログラム中に多数潜んでいる依存関係の抽出が可能になるといえる．

6.2　関連研究

Bergeron ら [4] は，Java のバイトコードを対象とする静的スライスについて述べており，エイリアス関係を考慮したデータ依存関係のグラフ (*A-DDG*) を定義し，A-DDG に基づくスライシングアルゴリズムを提案している．このアルゴリズムはバイトコード中に存在し得るエイリアス関係を考慮しているが，実装にまでは至ってない．本研究は手法の提案だけではなく，実装まで至っていることから有効なものと考えられる．

7.　まとめと今後の課題

本論文では，オブジェクト指向言語を対象とするエイリアス関係を考慮した静的フログラムスライス計算手法を提案した．提案した手法では，引数の参照渡し，参照変数，ホインタを介した間接参照などで生じるエイリアス関係を考慮することで，

表 3　スライスサイズの比較（単位：行）

フログラム	エイリアス考慮	エイリアス無視
P1	8	6
P2	42	30

従来の依存関係解析手法では発見することのできない依存関係を抽出できるようになり，スライス計算の精度が高まった．

また，提案手法をスライシングツールとして，我々が研究開発を行っている Java フログラム解析フレームワークに追加実装し，その有効性を検証した．

今後の課題として，

- スレッド，例外を含めた複雑な制御構造の解析への対応
- エイリアス関係，各種依存関係のデータベース化
- 大規模システムに対する適用実験

などが挙げられる．

文　献

[1] A.V.Aho, S.Sethi and J.D.Ullman: "Compilers : Principles, Techniques, and Tools" (1986).

[2] B.Steensgaard: "Points-to analysis in almost linear time", *In Proceedings of the 23rd Annual ACM SIGACT-SIGPLAN Symposium on the Principles of Programming Languages*, pp.32–41, Beach, Florida (1996).

[3] H.Agrawal and J.Horgan: "Dynamic Program Slicing", *SIGPLAN Notices*, vol.25, no.6, pp.246–256 (1990).

[4] J.Bergeron, M.Debbabi, M.Debbabi, M.M.Erhioui and B.Ktari: "Static Analysis of Binary Code to Isolate Malicious Behaviors", *In Proceedings of the IEEE 8th International Workshops on Enterprise Security*, pp.184–189, Stanford University, California, USA, June 16–18 (1999).

[5] J.Gosling, B.Joy and G.Steele: "The JAVA™ Language Specification" (1996).

[6] K.B.Gallagher and J.R.Lyle: "Using program slicing in software maintenance", *In IEEE Transactions on Software Enginnering*, vol.17, no.8, pp.751–761 (1991).

[7] M.Weiser: "Program Slicing", *Proceedings of the 5th International Conference on Software Engineering*, pp.439–449, San Diego, California (1981).

[8] P.Tonella, G.Antoniol, R.Fuitem and E.Merlo: "Flow Insensitive C++ Pointers and Polymorphism Analysis and its Application to Slicing", *Proceedings of the 19th International Conference on Software Engineering*, pp.433–443, Boston, Massachusetts (1997).

[9] R.P.Wilson and M.S.Lam: "Efficient context-sensitive pointer analysis for C programs", *Proceedings of SIGPLAN'95 Conference on Programming Language Design and Implementation*, pp.1–12, La Jolla, California (1995).

[10] S.Bates and S.Horwitz: "Incremental program testing using program dependence graphs", *Conference Record of the Twentieth ACM Symposium on Principles of Programming Languages*, pp.384–396, Charleston, South Carolina (1993).

[11] S.Horwitz and T.Reps: "The use of program dependence graphs in software engineering", *Proceedings of the 14th International Conference on Software Engineering*, pp.392–411, Melbourne, Australia (1992).

[12] 大畑 文明, 近藤 和弘, 井上 克郎: "エイリアスフローグラフを用いたオブジェクト指向フログラムのエイリアス解析手法", 電子情報通信学会論文誌, vol.J84-D-I, no.5, pp.1–11 (2001).

[13] 山中 祐介, 大畑 文明, 井上 克郎: "フログラム解析情報の XML データベース化 –提案と実現–", コンヒュータソフトウェア, vol.19, no.1, pp.39–43 (2002).