研究速報

エイリアス関係を考慮した Java スライシングツール

山中　祐介†　　横森　励士†（学生員）
井上　克郎††（正員）

Java Slicing Tool Using Alias Relation

Yuusuke YAMANAKA†, *Nonmember*,
Reishi YOKOMORI†, *Student Member*,
and Katsuro INOUE††, *Regular Member*

† 大阪大学大学院基礎工学研究科，豊中市
Graduate School of Engineering Science, Osaka University, 1-3 Machikaneyama-cho, Toyonaka-shi, 560–8531 Japan
†† 大阪大学大学院情報科学研究科，豊中市
Graduate School of Information Science and Technology, Osaka University, 1-3 Machikaneyama-cho, Toyonaka-shi, 560–8531 Japan

あらまし　複数の変数が同一空間を指すエイリアス関係の解析は，オブジェクト指向言語におけるスライス計算には必須である．本論文では，エイリアス関係を考慮した Java 向けの静的スライシングツールの実現について述べる．

キーワード　プログラムスライシング，エイリアス関係，静的解析，Java

1. ま え が き

近年の大規模化・複雑化するプログラムに対して，プログラムスライシング [2] がデバッグの効率化を目的として提案されている．プログラムスライシングとは，プログラム文中においてある文 s のある変数 v（スライス基準 $\langle s, v\rangle$ と呼ぶ）の値に影響を与える文の集合（プログラムスライス，以下スライス）を抽出する技術であり，プログラム文間の依存関係を解析することで求められる．これまでオブジェクト指向言語において，様々なプログラムスライシング手法の提案・実現がなされてきた [3]．オブジェクト指向言語の場合，参照変数を用いることが多いため，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が発生しやすい．しかし，既存の手法では十分に考慮されていないため，結果が正しくないなどの問題があり，考慮されていてもエイリアス関係の解析（エイリアス解析）についてあいまいである．

そこで，本論文では Java を対象としてエイリアス関係を考慮した静的依存解析手法を提案する．エイリアス関係を考慮することで，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，詳細なスライス計算が期待できる．また，提案手法をスライシングツールとして実装し，本手法の有効性について検証する．

2. プログラム依存グラフと依存関係の解析

プログラムスライスの計算には，プログラム文間の依存関係からプログラム依存グラフ（以下，PDG）を構築した上で，プログラム依存グラフ上でスライス基準に対応する節点からの到達可能節点を求める手法が一般的に用いられている [3], [4]．PDG 上の節点はプログラム上の文を表し，辺はプログラム文間の依存関係を表す．PDG 上の辺は，変数間のデータのやり取りの把握を目的としたデータ依存辺と，プログラム文間の制御構造のやり取りを目的とした制御依存辺の 2 種類の辺に大別される．

一般の依存関係解析手法では，メソッドや手続き単位での解析を，実行順を考慮しながら依存関係の変化が起こらなくなるまで繰り返し行う．メソッド間の呼出し関係は，呼び出される側のメソッド内で必要なメンバ変数や引き数に関する特殊節点を，呼出し側と呼び出される側双方に用意し，特殊節点間に辺を引くことで表現される [3], [4]．

メソッド内での依存関係解析では，プログラムの各実行時点（文）における到達定義集合（Reaching Definition Set）[1] を計算することで，プログラムの各実行時点（文）において参照される変数がどの文で定義されたかを識別する．到達定義集合は，(変数名，最後に定義された場所）を要素とした集合で，解析の際には，到達定義集合を変数表として管理し，プログラムの実行順に従って変数表の更新やデータ依存辺の抽出を行っていく手法が用いられる．しかしオブジェクト指向言語の場合，参照変数を用いることが多いため，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が発生しやすく，すべての実行経路を考慮した解析手法である静的解析を行う際に正しい結果が得られないという問題がある．

3. エイリアス関係を考慮した依存関係解析

提案手法では，エイリアス関係にある変数の組からなるエイリアス表を作成した上で，変数表を拡張することで，エイリアス関係を考慮した静的依存関係解析を行う．提案手法に基づく PDG 構築の手順を以下に示す．

Phase 1：　構文解析・意味解析を行い，プログラム内の各文で定義・参照されている変数を把握する．

Phase 2：　各インスタンス生成式及びメインメソッドの引数とエイリアス関係にある変数の組を求め，エイリアス表を作成する．

Phase 3：　PDG の節点を作成し，実行制御構造から

制御依存辺を求める．

Phase 4： プログラム中の全メソッドを依存関係の変化が起こらなくなるまで実行順を考慮しながら繰り返し解析を行う．提案手法では，静的初期化ブロックやコンストラクタなども，メソッドと同等にみなす．

3.1 エイリアス表について

エイリアス関係とは，同じメモリ空間を指す可能性がある式間の同値関係を指す．提案手法では, [5] で提案されている手法を用いて，プログラム内に存在するそれぞれのインスタンス生成式及びメインメソッドの引き数に対して，その式とエイリアス関係にあるオブジェクトへの参照式（参照変数，参照変数を返すメソッド呼出し文）の集合を求める．インスタンス生成式により宣言されたオブジェクトのメモリ領域が確保されるため，インスタンス生成式それぞれに対して異なる ID を割り振り，エイリアス表として保存する．

エイリアス表は，ID と,（メソッド・行番号・変数名）で表される参照式の集合で構成されている．メソッド内の依存関係解析時に参照変数が定義された際に，どのメモリ領域を指し得るかをエイリアス表を用いて判定することで，どの参照変数が各実行時点においてどのメモリ領域を参照し得るかを把握できる．図 1 に，サンプルプログラムにおけるエイリアス表を示す．

3.2 メソッド内の依存関係解析について

提案手法では，既存の手法と同様に実行順に従って変数表を更新することで，メソッド内のデータ依存辺を抽出する．しかし，参照変数に関するエイリアスを考慮するために，変数表を拡張し，メンバ変数表を追加している．以下では，変数表の拡張及びメンバ変数表について述べた上で，メソッド内の解析の概略を説明する．

[変数表の拡張] 提案手法では，同じオブジェクトが複数生成された場合を考慮して，オブジェクトを生成元（インスタンス生成式）ごとに区別した上で参照変数の解析を行う．解析においては，変数表における参照変数に関する要素を（変数名，オブジェクトが参照し得る ID，最後に定義された場所）と拡張することで，どのインスタンス化式で生成されたオブジェクトを参照し得るかという情報を付帯する．

[メンバ変数表] 提案手法では，メソッド内で利用されるオブジェクト内に存在するメンバ変数を生成元ごとに区別するため，メンバ変数表を用意する．メンバ変数表の要素は（ID，メンバ変数名，最後に定義された場所，メンバ変数が参照し得る ID（メンバ変数が参照変数である場合））で表される．これにより，複数の参照変数を用いてある一つオブジェクトのメンバ変数が操作されたとしても，依存関係の抽出が可能となる．

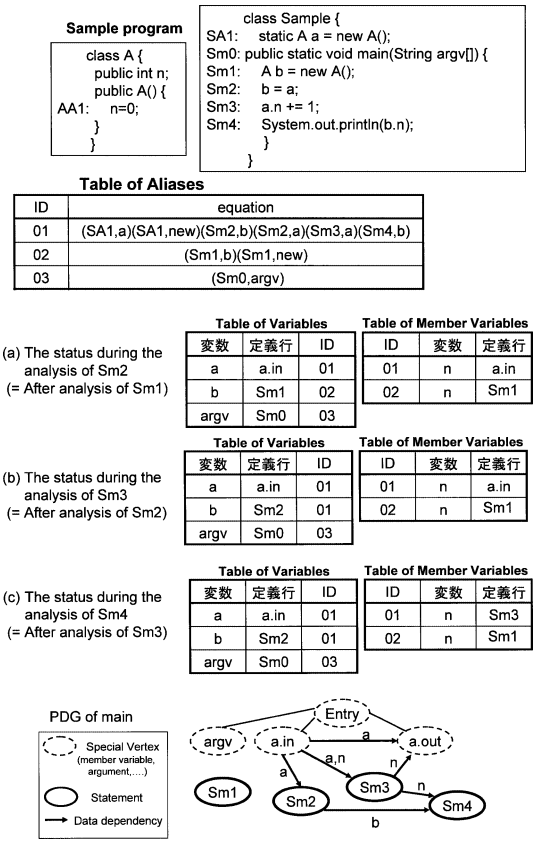

図 1 提案手法の概略
Fig. 1 The outline of the proposal technique.

Step 1：変数表及びメンバ変数表の作成

メソッド内の解析においては前処理として，まずメソッドの引き数及び，そのメソッドが属するクラスのメンバ変数から変数表及びメンバ変数表を作成する．

- 引き数が存在する場合，メソッドの宣言部を定義行とみなして変数表に登録する．引き数に参照型の変数を利用した場合，エイリアス表を参照してどの ID を参照し得るかを解析し，その ID を変数表に登録する．また，引き数のオブジェクト中に参照可能なメンバ変数があれば，そのメンバ変数をメンバ変数表に登録する．
- メソッドが属するクラスのメンバ変数は，特殊節点を定義行とみなして，定義行のみを変数表に登録する．参照型のメンバ変数の場合，メソッド中で初め

て参照された時点でエイリアス表を参照し，その ID を変数表に登録する．また，参照型のメンバ変数が指すオブジェクト中に参照可能なメンバ変数があるならば，そのメンバ変数をメンバ変数表に登録する．

Step 2：文の解析

実行順に従いメソッド内の文を先頭から解析していき，変数表を更新しながら，データ依存辺を抽出する．解析の概略を以下に示す．

Case 1：解析中の文において変数が宣言された場合，変数名だけを変数表に登録する．

Case 2：解析中の文において変数の定義が行われている場合，対応する変数表の定義行を解析中の行に更新する．参照型の変数が定義されている場合，エイリアス表を参照してどの ID をもち得るかを解析し変数表に登録する．インスタンス化式，オブジェクトを返すメソッドによって参照変数が定義された場合，返されたオブジェクト中に参照可能なメンバ変数があれば，そのメンバ変数を ID 及び定義行とともにメンバ変数表に登録する．

解析例：図 1 の Sample クラスの main メソッドの 2 行目では，変数 b が定義されている．b は参照型の変数であるため，エイリアス表を参照したところ，ID が 01 の領域を参照していることがわかる．そのため，2 行目の解析において (a) の変数表が (b) の変数表に更新される．

Case 3：解析中の文において変数が参照されている場合，変数表中の対応する変数の定義行から解析中の行へデータ依存関係が存在するとみなし，データ依存辺を引く．

解析例：図 1 の Sample クラスの main メソッドの 2 行目では，変数 a が参照されている．2 行目の解析時の変数表である (a) の変数表を用いて変数 a が特殊節点 a.in で定義されたことを把握し，a.in から 2 行目にデータ依存辺を引く．

Case 4：参照変数のメンバ変数を参照している場合，参照変数の ID を変数表を用いて特定した上で，対応するメンバ変数表の変数の定義行から解析中の行へデータ依存関係が存在するとみなし，データ依存辺を引く．

解析例：図 1 の Sample クラスの main メソッドの 4 行目では，変数 b.n が参照されている．4 行目の解析時の変数表である (c) の変数表を用いることで，変数 b が ID が 01 の領域を指していることがわかる．メンバ変数表で ID が 01 である変数 n の定義行を参照し，3 行目から 4 行目へデータ依存辺を引く．

Case 5：If 文，while 文などのように制御構造中で分岐が存在する場合，変数表及びメンバ変数表を複製し，それぞれの実行経路に対して解析を行う．また，実行経路が合流する地点では，それぞれの実行経路における表を合成する．また繰返し文の場合，繰返し地点において各表が変化しなくなるまで文の解析を繰り返す．

Case 6：メソッド呼出し文では，メソッドの呼出し中に定義された変数に関して変数表及びメンバ変数表を更新する．

Step 3：解析後の処理

メソッド内の文を最後まで解析した後には，メソッド呼出し実行後の状態を保持するための特殊節点に対して変数表を用いてデータ依存辺を引く．その後に，メソッドが属するクラスのメンバ変数や変数渡しの引き数の値が更新された場合，呼出し元の特殊節点に変数が定義されたことを記録する．

3.3 エイリアスを考慮した依存関係解析の例

提案手法では，既存の手法における依存関係解析手法に，参照変数に関する処理を追加することでエイリアス関係を考慮した解析を行っている．図 1 の Sample クラスの main メソッドの 4 行目の解析を行う場合を考える．4 行目では変数 b とそのメンバ変数である n が参照されており，b と n に関してデータ依存関係が存在する．変数 b は 4 行目の解析時の変数表である (c) の変数表より，2 行目で定義されていることがわかるため，変数 b に関して 2 行目から 4 行目へのデータ依存関係を抽出できる．しかし，メンバ変数 n への処理を行うためには変数 n がどのオブジェクト上の n であるかの判別が必要である．

提案手法では，変数表中で参照変数が参照し得るオブジェクトを ID として保存することで，どのメンバ変数表を参照すべきであるかを特定できる．この場合，b は ID が 01 である領域を参照していることがわかるため，ID が 01 のオブジェクトのメンバ変数 n を参照することで 3 行目から 4 行目へのデータ依存関係を抽出できる．このように参照し得るオブジェクトの特定を行うことで，テキスト情報だけでは判断しがたいエイリアス関係を考慮した依存関係の抽出が可能となる．

4. スライシングツールの実現

我々の研究グループでは，Java プログラム解析フレームワーク（JAF）に関する研究を行っている．JAF は意味解析木を構築する解析木構築部，構文木情報

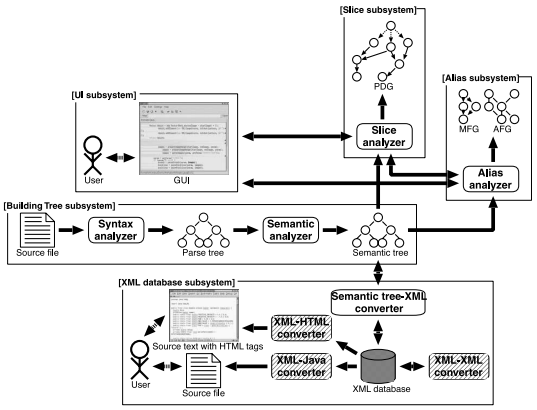

図 2　Java プログラム解析フレームワーク（JAF）
Fig. 2　Java program analysis framework (JAF).

などを XML データベース化する XML データベース部 [6]，エイリアス解析を行うエイリアス解析部 [5]，UI 部などの Java プログラムの静的解析を行うためのライブラリ，ツール群から構成されている．

提案手法の実現においては，この JAF にスライス計算部を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試作した．スライシングツールを含めた JAF を図 2 に示す．スライス計算ライブラリでは，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関係情報の抽出をしながら PDG 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してスライスを計算し，結果を表示する．

4.1 評　　価

実装したスライシングツールを用いて解析コスト及びスライスの正確性について評価実験を行った．対象プログラムの概要は，実験用に作成したプログラム（P1，クラス数 272（内 JDK は 270）），簡易ドローツール（P2，クラス数 876（内 JDK は 873））である．解析においては，JDK を含む，すべての参照しているソースコードすべてに対して PDG を構築している．

- 解析コスト

解析時間，メモリ使用量の結果を表 1 に示す．現在の計算機環境で，十分解析を行うことが可能であるといえるが，効率化のため JDK を PDG 構築の対象に含めるかの選択が必要になると考えられる．

- スライスの正確性

エイリアス関係を考慮した場合としない場合とで，同じスライス基準に対してスライス計算を行った．得られたスライスサイズの違いを表 2 に示す．エイリアス関係を考慮した場合は，考慮しない場合に比べてスライスサイズが大きい．しかし，いずれの場合もエイリアス関係にある参照変数がもたらした依存関係が存在するため，考慮しない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，正確なスライス計算を行う上で提案手法は必須であるといえる．

表 1　解析コスト
Table 1　Analysis cost.

Program	Analysis time (ms)	Used memory (MB)
P1	17,206	149
P2	67,791	457

表 2　スライスサイズの比較（単位：行）
Table 2　Comparison of slice size.

Program	Use alias	Not use alias
P1	8	6
P2	42	30

5. む す び

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プログラムスライス計算手法を提案し，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実現した．提案手法により，正確で実用的なスライス計算が可能になる．

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，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, Addison-Wesley, 1986.

[2] M. Weiser, “Program slicing,” Proc. 5th International Conference on Software Engineering, pp.439–449, San Diego, California, 1981.

[3] D. Liang and M.J. Harrold, “Slicing objects using system dependence graphs,” Proc. International Conference on Software Maintenance, pp.358–367, Bethesda, MD, 1998.

[4] S. Horwitz and T. Reps, “The use of program dependence graphs in software engineering,” Proc. 14th International Conference on Software Engineering, pp.392–411, Melbourne, Australia, 1992.

[5] 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プログラムのエイリアス解析手法，” 信学論（D-I），vol.J84-D-I, no.5, pp.443–453, May 2001.

[6] 山中祐介，大畑文明，井上克郎，“プログラム解析情報の XML データベース化—提案と実現，” コンピュータソフトウェア，vol.19, no.1, pp.39–43, 2002.

（平成 15 年 4 月 3 日受付，6 月 11 日再受付）