特別研究報告

題目

JAVAプログラムのエイリアス情報表示ツールの試作

指導教官
井上 克郎 教授

報告者
近藤 和弘

平成 12 年 2 月 23 日

大阪大学 基礎工学部 情報科学科

平成 12 年度 特別研究報告

JAVA プログラムのエイリアス情報表示ツールの試作

近藤 和弘

内容梗概

エイリアスとは，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，識別子が異なるが同じメモリ領域を指す可能性のある変数および式の集合である．オブジェクト指向言語の 1 つである Java は C++とは異なり，ポインタはなく参照変数のみ存在する．そのため解析結果が直接プログラム理解に結びつきやすく，プログラムスライス・コンパイラ最適化のためだけでなく．デバッグや保守においてもエイリアス解析の利用が期待できる．

我々の研究グループでは，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 および オブジェクト指向言語に対するその拡張 の提案を行い，提案手法を Java を対象として実装を行ったが，エイリアスの解析結果は，変数および式の集合であるため，その利用にはエイリアス情報の視覚化が求められる．

本研究では，エイリアス解析結果を有効に利用するための，エイリアス表示ツールの試作を行った．本ツールは，解析部から渡されたエイリアス解析結果を利用して，エイリアス集合の各要素ごとの詳細な情報をソースコードと対応させながら表示でき，エイリアス情報を有効に利用するための様々な機能を有するため，エイリアス解析による，プログラム理解，デバッグなどの支援が可能となる．

主な用語

エイリアス (Alias)
視覚化 (visualization)
Java

目 次

1 まえがき

プログラムのデバッグ支援に有効な方法として**プログラムスライス** *(Prgram Slice)*[11] がある．プログラムスライスとは，プログラム P 中のある文 s に対して，s で参照しているある変数 v の内容に影響を与えうる文の集合 $\mathcal{Q}$ のことである．現在では, デバッグ支援だけでなくテスト，保守，プログラム合成，プログラム理解などにも利用されている. このプログラムスライスの計算は，

Phase 1: データ依存解析

Phase 2: 制御依存解析

Phase 3: プログラム依存グラフ (PDG) の構築

Phase 4: PDG を利用しスライスを計算

という過程をたどるが，**Phase 1: データ依存解析**においては，各文でどの変数が定義・使用されているかが判明していなければならない．そのため，ポインタ (参照) が存在するプログラミング言語のデータ依存解析では，エイリアスの解析が前提となっている．

さらに，オブジェクト指向言語の 1 つである JAVA[4] は C++[10] とは異なり，ポインタはなく参照変数のみ存在する．そのため解析結果が直接プログラム理解に結びつきやすく，プログラムスライス・コンパイラ最適化のためだけでなく．デバッグや保守においてもエイリアス解析の利用が期待できる．

そこで我々の研究グループでは，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 および オブジェクト指向言語に対するその拡張 の提案を行い，提案手法を JAVA を対象として実装を行った [12].

このエイリアスの解析結果はエイリアス集合 (2 節参照) であり，あくまで変数および式の集合である．したがって，その解析結果を直接利用することは困難であるため，本研究では，エイリアス解析結果を有効に利用するための，エイリアス表示ツールの試作を行った. 本ツールは，解析部から渡されたエイリアス解析結果を利用して，エイリアス集合の各要素ごとの詳細な情報を，ソースコードと対応させながら表示できるため，エイリアス解析による，プログラム理解，デバッグなどの支援が可能となる．また，本ツールで用いられた表示手法は，プログラミングスライスの計算結果の表示にも応用できるものである.

以降，2 節ではエイリアスについて簡単に紹介する．3 節では，プログラムビジュアライゼーションとエイリアス情報表示の参考となるスライスの表示について紹介する．4 節でツールの持つ各機能の詳細な設計と説明を行ない， その機能をツールに実装した場合の具体的な使用例を 5 節で紹介する．7 節でまとめと今後の課題について述べる.

2　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，識別子は異なるが同じメモリ領域 (オブジェクト) を指す変数および式の集合 (以降，単に**エイリアス集合** *(Alias Set)* と略す) である．

エイリアス解析の利用分野としては，まえがきで述べたプログラムスライスのほかに，コンパイラの最適化技法がある．

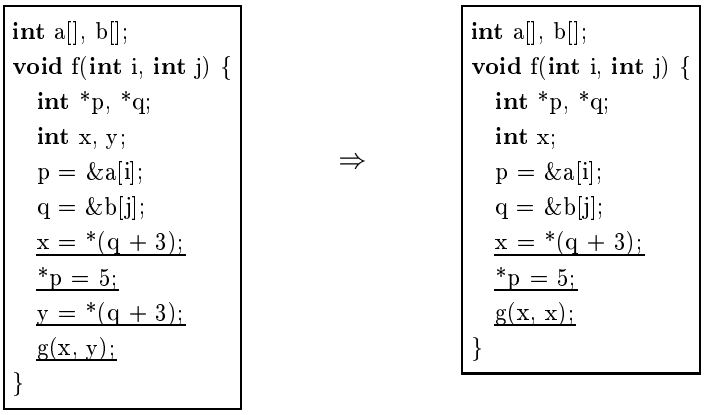


図 1: コンパイラの最適化の例

図 1 は C 言語で書かれたプログラム例であるが，解析によりポインタ p・q はエイリアスを生成しないと判断でき，文 y = *(q + 3) の省略，および，変数 y の変数 x への置き換えが可能となる．

エイリアス集合を導き出す**エイリアス解析** *(Alias Analysis)* 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 2 つに分けることができる．以下，FS 解析・FI 解析を簡単に説明する．

FS エイリアス解析

FS エイリアス解析とは，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う．一般に**到達エイリアス集合** *(Reaching Alias Set)* を利用して解析を行う．図 2 に変数 c(太枠部) の FS エイリアス (網掛部) を示す．

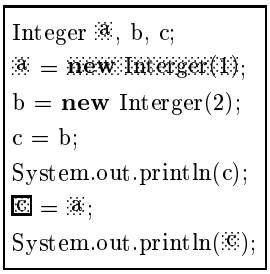


図 2: FS エイリアス解析を行った結果

FI エイリアス解析

FI エイリアス解析とは，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う．一般にエイリアス (または，Point-to) グラフを利用して解析を行う．図 3 に変数 c の FI エイリアスを示す．

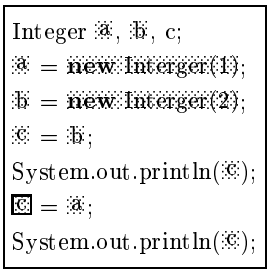


図 3: FI エイリアス解析を行った結果

FS 解析と FI 解析の比較

正確性: FS 解析はプログラム文の実行順を考慮しているため，FI 解析よりも正確性は高い

> **正確性** *(accuracy)* とは,「少なくとも実在するエイリアス集合は含まれているとして，どれだけ実在しないエイリアス集合を取り除くことができるかの程度」を表す.

コスト: FS 解析は FI 解析よりも時間計算量・空間計算量を必要とする

文献 [5] において，両者の詳細な比較がなされている.

Java とエイリアス

オブジェクト指向言語の 1 つである JAVA は C++とは異なり，ポインタはなく参照変数のみ存在する．そのため解析結果が直接プログラム理解に結びつきやすく，プログラムスライス・コンパイラ最適化のバックエンド以外の利用が期待できる．

図 4 に JAVA プログラムとその実行結果を示す．ユーザは文 A.print() の出力異常を認識し，参照変数 A に関するエイリアスを抽出を試みる．網掛部が文 A.print() の A に関するエイリアス，下線部がそのエイリアスが呼び出したメソッドである．この場合，文 e.add_salary(50) を add_salary(50) に変更することで期待動作を行うようになる．

```
class Employee {
  String name; int salary; Employee boss;
  Employee(String n, int s) {
    name = n; salary = s; boss = null;
  }
  void add_salary(int n) {salary += n; }
  void set_boss(Employee e) { boss = e; }
  void print() {
    System.out.println(name + " Salary:" + salary);
  }
}
class Manager extends Employee {
  Manager(String n, int s) {super(n, s);}
  void manage(Employee e) {
    e.set_boss(this); e.add_salary(50);
  }
}
class Office {
  public static void main(String args[]) {
    Employee A = new Employee("Mr.A", 700)
    Manager B = new Manager("Mr.B", 850);
    B.manage(A); A.print(); B.print();
  }
}
```

```
% java Office
Mr.A Salary: 750
Mr.B Salary: 850
```

図 4: JAVA プログラム

3　エイリアス情報の視覚化

プログラムビジュアライゼーション

プログラム開発を支援するために，プログラムに関する情報を視覚的に表現することをプログラムビジュアライゼーションと呼ぶ．また，最近では単にプログラムに関する情報だけでなく，バージョン情報，モジュール情報，プロジェクト情報等，ソフトウェア開発に関する情報一般を視覚化することを ソフトウェアビジュアライゼーションと呼ぶ．
プログラムビジュアライゼーションは，情報検索，データ解析，プログラムデバッグなどに幅広く利用されており，必要な技法として次のようなものが挙げられている [19]．

- 必要なもののみ表示する技法
- 全体と詳細を同時に見る方法
- 抽象的データの画面へのマッピング法
- 自動レイアウト手法
- 操作しやすいインタフェース

また，具体的な表示技法として次のようなものが挙げられる．

- 自動配置
- 3 次元視覚化
- ズーミング (線型/非線型)
- 色の濃淡/透明度
- アニメーション

実現例の一部を以下に紹介しておく．

- 階層構造の視覚化: Corn Tree[3]
  あるレベルの子ノードを，親ノードを頂点とする円錐の底辺周囲部に配置することで，木を 3 次元表示している．ディレクトリやバージョンの管理など階層構造の視覚化に積極的に利用されている．
- アルゴリズムアニメーション: Zeus[2]
  アルゴリズムアニメーショ ンに後述する音や３次元の概念を実験的に導入している．また１つの実体に対して，複数の図形やテキストを対応さ せ編集できる Multiview Editing の機能が実現されている．

- 局所的詳細と大局的概略の統合表示: Perspective Wall[6]
  画面中央部分は通常通り表示し，端の部分になると奥行き方向に沈みこんでいくような表示にすることで，中央部では完全な情報が観察でき，周辺部では概略のみがみえるようにする．このように全体の特徴 (Context) を把握しながら，ユーザに興味のある視点 (Focus) 付近は詳細に情報を観察できるようなアプローチは **Focus&Context** と呼ばれている．

- オブジェクト指向プログラミングへの応用: クラスライブラリの視覚化 [13]
  クラス階層とメソッド継承関係が一つの 3 次元表示の中に組み込まれている．クラスに定義されているメソッドはそれぞれ対応するクラスの奥行き方向に並べて表示されるが，同一のメソッドは同じ z 軸の位置になるように配置される．これによってメソッドのオーバーライドなどといった関係が一目瞭然となる．

- インタラクティブな視覚化: 納豆ビュー [14]
  3 次元空間に置かれた情報の中のある要素を”つまみあげる”という操作が提供されている．納豆の名前通り，関連する情報ノードが糸を引くように一緒に持ち上がってくるのが特徴である．このように単に表示するだけでなく，**対話性**を持ち込むことで問題を解決しようとしているものもある．

プログラムビジュアライゼーションは**プログラムスライス** *(Prgram Slice)*[11] の分野においても利用されており，例としては，SeeSoft[9] をベースにして AT&T で開発された SeeSlice[1] などががある．SeeSlice で用いられている表示方法には，次のようなものがある．

- プログラムを三階層 (ファイル，手続き，文) に分け，階層別に管理して表示
- 依存グラフ上でのスライス基準からの距離に応じた強調表示
- スライスを含まない手続きに関する情報の非表示
- スライスの全体像を表示するウインドウ
- 局部的なソースコードを表示するウインドウ

これらの手法はスライスの表示方法として，上記のビジュアライゼーションに必要な技法のうち,「必要なものみ表示する技法」と「全体と詳細を同時に見る方法」において特に優れているため，エイリアスの表示においても応用できる部分を多く含んでいると思われる．

ところが，SeeSlice で用いられている手法は，手続き型言語を対象としているが，今回エイリアス情報の表示を行なう対象となる言語はオブジェクト指向言語の Java である．そのため，オブジェクト指向の特徴を生かしながら表示することが求められる．

以上より，本研究では次のような目的を持って、解析部から渡されたエイリアス情報の表示 (図 5 の GUI 部の作成) を行なった。

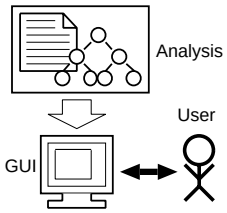


図 5: システム構成図 (略図)

目的 1: エイリアス集合の効果的な表示 (「必要なもののみ表示する技法」の実現)

目的 2: エイリアス集合全体像の把握と理解の簡易化 (「全体と詳細を同時に見る方法」の実現)

これらの目的を実現するための機能を 4 で説明し、その機能を実際に使用した例を画面写真とともに 5 で紹介する。

4 機能設計

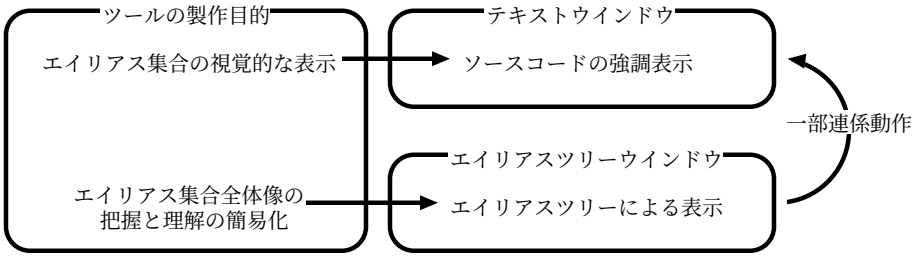


図 6: エイリアス情報表示ツールの製作目的と実現方法

エイリアスの解析結果を表示するにあたって，大きく分けて次の二つの手法を用いた．

手法 1: ソースコードの強調表示

手法 2: エイリアスツリーによる表示

前節でのエイリアス情報表示の目的 1 を実現するものが手法 1，目的 2 を実現するものが手法 2 である．また，この二つの手法はそれぞれ独立したものであるが，一部連係動作 (4.2 の「テキストとの対応」参照) をする．手法 1 はテキストウインドウ (メインウインドウ) 上で，手法 2 はエイリアスツリーウインドウ上で実装される．図 6 に，エイリアス情報表示ツールの製作目的と実現方法の関連を示す．

以下では，それぞれの手法別に機能の説明を行う (具体的な表示例は 5 節を参照のこと)．

4.1 テキスト上での強調表示

テキスト上での強調表示は，プログラムのソース中に含まれるエイリアスに，いかに着目させるか，ということを目指して行なった．これは，次の二つの表示の組合せによって行われる．

- ソース中でエイリアス集合に含まれる部分 (**エイリアス部分**と呼ぶ) の表示
- ソース中でエイリアス集合に含まれない部分 (**非エイリアス部分**と呼ぶ) の表示

ソース中のエイリアス部分はより強調し，非エイリアス部分は最低限の情報を残した上で簡略表示することで，抽出されたエイリアス集合の把握が容易となる．ソースコード中のエイリアス部分と非エイリアス部分の例を図 7 に示す．

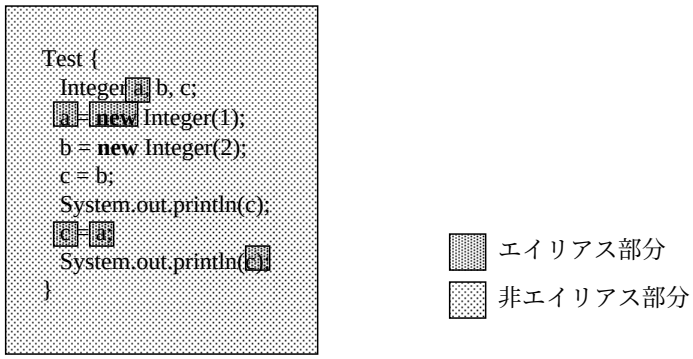


図 7: ソースコード中のエイリアス部分と非エイリアス部分

4.1.1 ソース中のエイリアス部分の表示

エイリアス部分は，エイリアス解析の直接の結果でもあり，ユーザが着目したい部分である．そのために強調表示を行ったが，具体的な強調方法として以下のような方法を用いた．

- **背景色の変更**
  ソースを表示しているテキストウインドウの背景色は通常白であるが，ソース中のエイリアス部分の背景色に色を挿入することで，ソースから浮きあがらせることができる．

- **フォントの変更**
  通常テキストでソースの表示に使用されているフォントよりも，やや大きめのフォントを用いることで，ソース中のエイリアス部分を目立たせ，かつ全体のバランスを崩さず表示する．

- **エイリアス集合別の背景色**
  複数のエイリアス集合を同時に表示させることが可能なため，エイリアス集合ごとに異なった背景色を用いることで，どのエイリアス集合に属するものか容易に区別できる．

4.1.2 ソース中の非エイリアス部分の表示

非エイリアス部分は，ソース全体と共にエイリアス部分を見渡したい場合などには必要であるが，エイリアス部分のみに着目したい場合には不必要な部分であるとも言える．したがって，状況に応じて非エイリアス部分の表示方法を変更できるように，以下のような方法から選択できるようにした．

- **縮小**

- **可変**
  エイリアス部分が存在する行に，近い行ほど大きいフォントで，遠い行ほど小さいフォントで表示する．これにより，全体の大まかな制御構造を把握しつつ，エイリアス部分に着目できる．
- **線分化**
  エイリアス部分が存在しない行は，一行が線になるまで高さを圧縮して表示する．ソースコードのサイズが大幅に小さくなるため，ソースコード中に存在するエイリアスの確認が容易になり，エイリアス部分だけに着目したい場合に有用である．また，線分化はインデント，行の長さは維持したまま行なわれるため，線分化されている非エイリアス部分のサイズなどは把握できる．

- **保存**
  エイリアスをとる前の，ソースコードの状態をそのまま保存する．ソースコード全体を見ながら，エイリアスを表示したい場合に用いる．また，縮小で表示を行なった後にこれを用いると，縮小された部分を保存したまま新たなエイリアスを表示することもできる．

4.1.3 視界の拡大

前述のように，非エイリアス部分の表示を「線分化」にすると，そのエイリアスに関して余計な部分が表示されないので，エイリアスを含む行だけに視点を集中できる．しかし,「線分化」では，エイリアス部分が存在する行のみのソースが表示されるので，エイリアス部分の前後で行なわれている処理が全く見えない．このような場合でも，再びソースコード全体を表示することなく，興味のある部分だけ前後のソースを見ることができる．つまり，エイリアス部分のある箇所に特定して，その付近に視界を広げることができる (図 8) わけである．視界の拡大は次のような単位で行うことができる．

前方： もう一行分上に視界を広げる

後方： もう一行分下に視界を広げる

前後： もう一行分上下に視界を広げる

おのおのの視界の拡大は連続して行うことができるため，必要な場合は次々とコードを表示していくことができる．具体的な操作については 5 節で述べる．

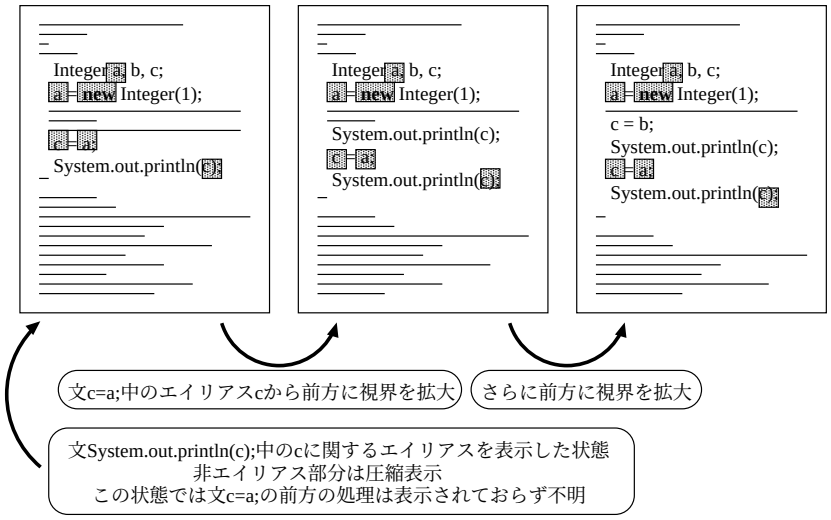


図 8: 視界の拡大

4.2 エイリアスツリーによる表示

エイリアス部分は，多数ののメソッド中に点在していたり，一つのファイルだけでなく複数のファイルにわたって存在することもある．したがって，テキスト表示だけでは，エイリアス部分がどのように分布しているかを把握しにくい．そのため，エイリアス集合の全体像を容易に把握できる機能が必要となる．それがこのエイリアスツリーである．エイリアスツリーの構造は図 9 のようになっており，後述の情報表示用リストと共にエイリアスツリーウインドウ上に表示される．また，ここではエイリアスの解析対象となる式を**エイリアス指定**と呼ぶ.
エイリアスツリーウインドウの機能を以下に示す.

4.2.1 情報表示

エイリアスツリーウインドウ上のエイリアスツリーで，ノードを選択すると，そのノードの情報が情報表示用リストに表示される．選択したノードによって，表示される情報の数は異なる．図 9 からもわかるように，ノードは 4 種類あり，表示される情報は次のようになっている.

- ノードの種類
  - エイリアス指定 (**エイリアス指定ノード** と呼ぶ)
  - クラス名 (**クラスノード** と呼ぶ)

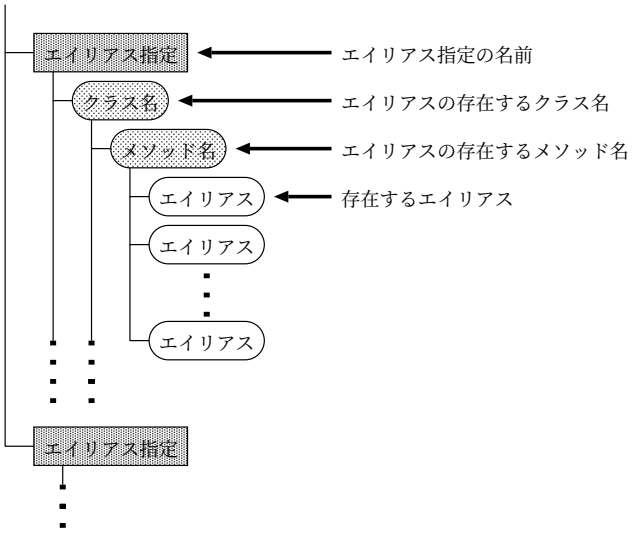


図 9: エイリアスツリーの構成

  - メソッド名 (**メソッド ノード** と呼ぶ)
  - エイリアス (**エイリアスノード** と呼ぶ)

- 共通に表示される情報
  - Name : 識別子
  - File : 存在するファイル名
  - Line : 存在する行
  - Column : 存在する桁
- 必要な場合は表示される情報
  - Class : クラス
  - Kind : 変数の種類
  - Modifier : 修飾子

4.2.2 テキストとの対応

エイリアスツリーにおいてノードが選択されると，前述のように情報をリストに表示すると同時に，テキストウインドウ上で，ソースコードの対応する場所にカーソルを移動させ，ズームアップして表示する (ソース中の非エイリアス部分の表示が，縮小または圧縮であっ

た場合は，もとの大きさに戻してからズームアップして表示する)．つまり，エイリアスツリーでノードが選択されると以下のような処理が行われることになる．

1: 選択されたノードに関する情報をリストに表示する

2: テキストウインドウ上のソースコード中の対応箇所にカーソルを移動させる

3: ズームアップ表示を行う

これにより，エイリアスツリーウインドウとテキストウインドウが対応するので，エイリアスツリーウインドウの情報の効果的な利用が可能となる．

4.2.3 エイリアスツリーからのファイルオープン

選択されたノードの存在するファイルが，まだオープンされていないファイルであった場合，次のような処理が行われる．

1: 選択されたノードに関する情報をリストに表示する

2: 新たにファイルをオープンする

3: そのファイル中に存在するエイリアス部分を強調表示する

4: 非エイリアス部分を現在選択されている方法で表示する

5: テキストウインドウ上のソースコード中の対応箇所にカーソルを移動させる

6: ズームアップ表示を行う

3，4 は実際には並行して行われる．非エイリアス部分の表示はファイルごとに指定できるが，エイリアスツリーからオープンした時点では，最近指定された表示方法で表示する．一連の処理の流れを図 10 に示す．
また，メニューの「ファイル」→「オープン」からのファイルオープンでは，エイリアス集合の要素の強調表示は行わない．

4.2.4 エイリアスツリーの整理

抽出して表示したエイリアスを消去せずに放置しておくと，エイリアスツリーの数も増え，エイリアスツリーウインドウが繁雑になってくる．そのような場合はエイリアスツリーウインドウの整理を行うことができる．整理を行うと現在存在する全てのエイリアスツリーの表示がエイリアス指定ノードのみの表示となるが，この状態でも各エイリアス指定ノードをクリックすると，エイリアス指定の情報が共に表示されるため，何についてのエイリアスであるかを見失うことはない．

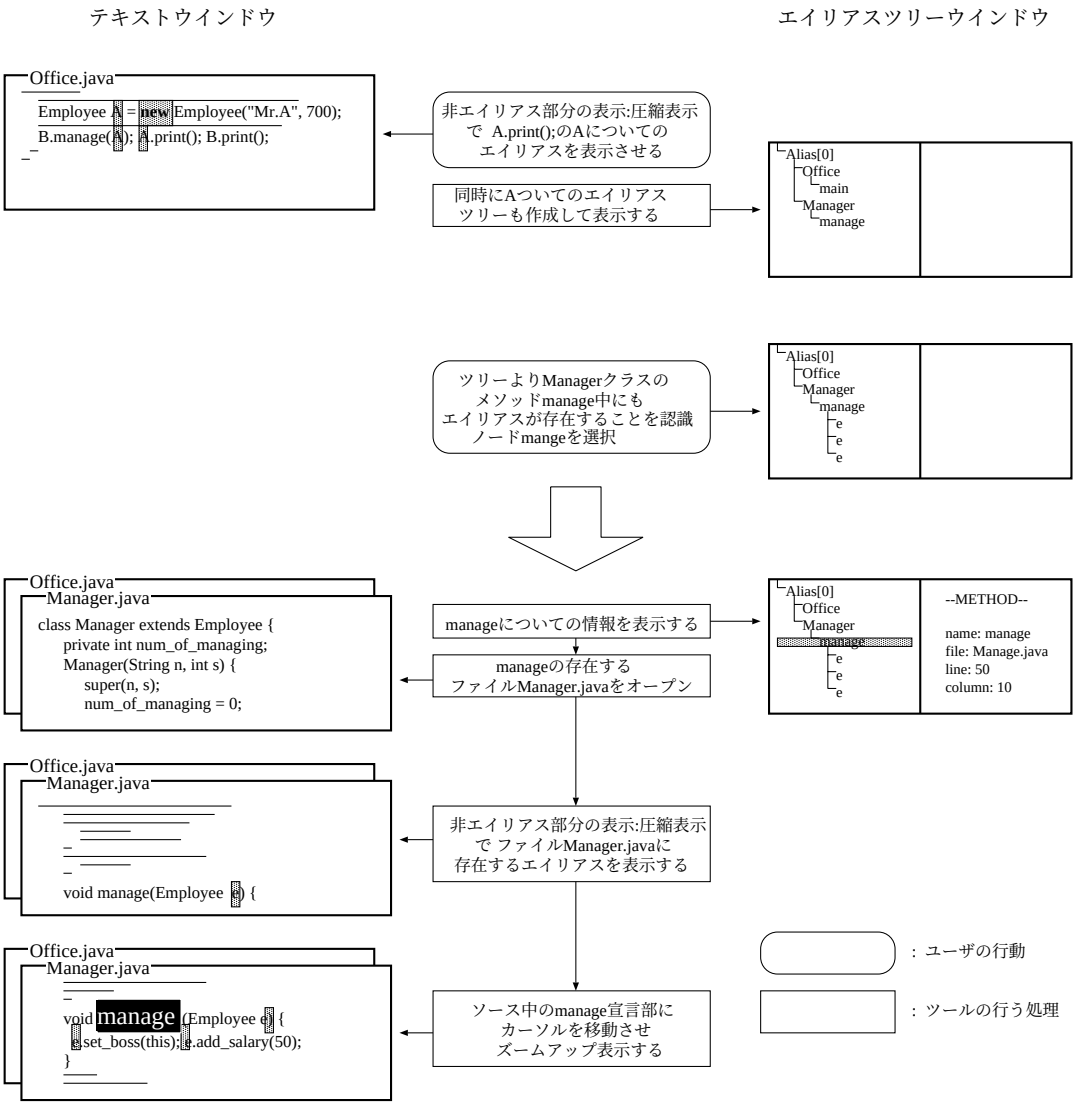


図 10: テキストとエイリアスツリーの連係例

4.3 オブジェクト情報ダイアログ

エイリアス解析とは直接関係はないが，プログラム理解の補助としてオブジェクト情報ダイアログという機能を作成した．これは，ソース中の各オブジェクトに対して，次のような情報を表示することができるものである．

- **オブジェクトのクラス**
- **オブジェクトの生成場所情報**
  - 生成された場所のファイル名
  - 生成された場所のクラス名
  - 生成された場所のメソッド名
  - 生成された行
- **オブジェクトの現在位置情報**
  - 現在位置のファイル名
  - 現在位置のクラス名
  - 現在位置のメソッド名
  - 現在位置の行

また，上の計 8 箇所に対応するソース上の位置 (ファイル名ならそのファイルの先頭，クラス・メソッド名ならその宣言部) に瞬時にカーソルを移動させることもできる．

5　実装ツールの機能

5.1　ツール概要

ここでは，前節で述べた機能をツールの実行画面と対応させながら説明する．ツールは C++[7] で記述されており，GUI ライブラリとして Gtk−−[20](GTK+[21]) を使用した．また，解析対象プログラムには，JDK 付属のサンプルコード Animator.java を用いた．構成図を図 11 に示しておく．

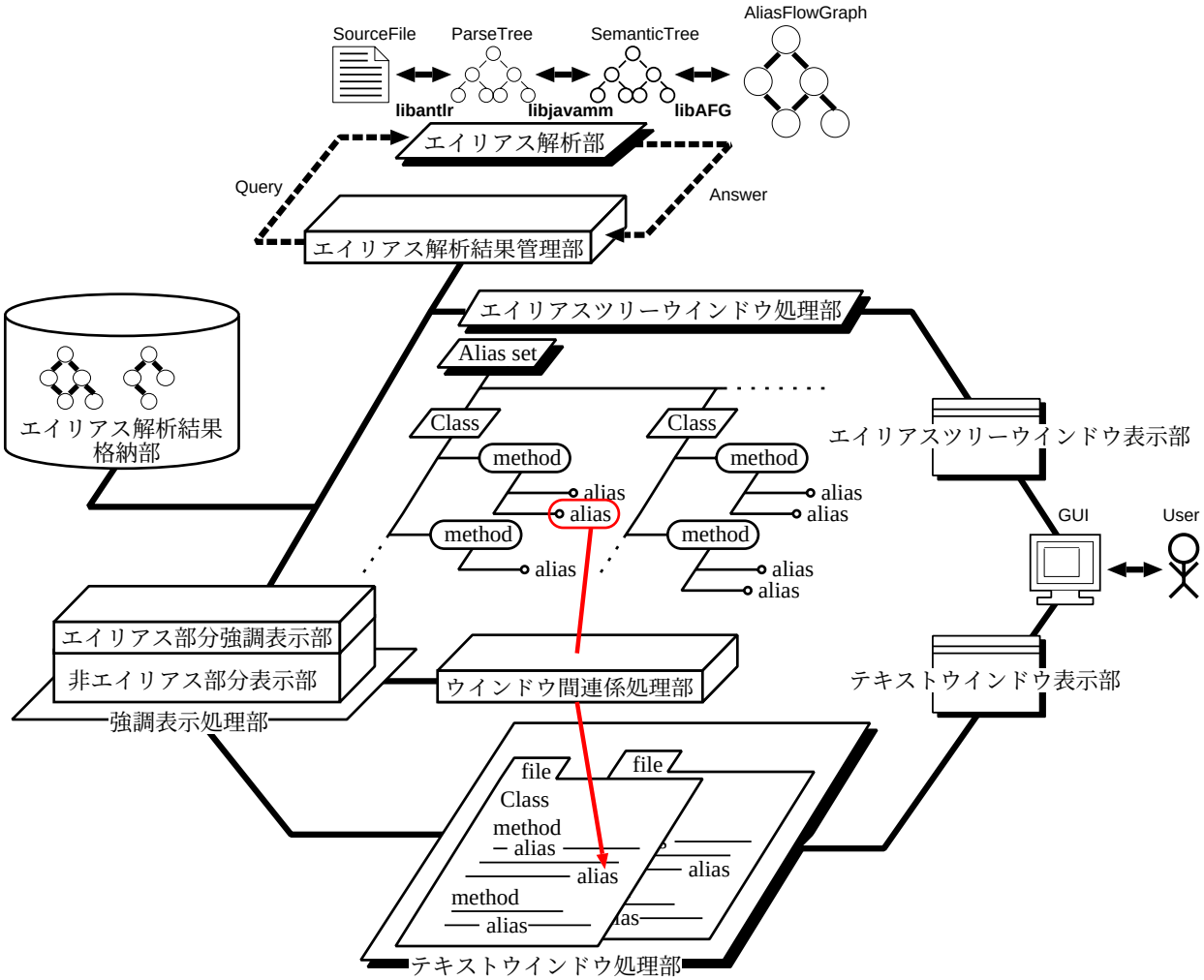


図 11: システム構成図

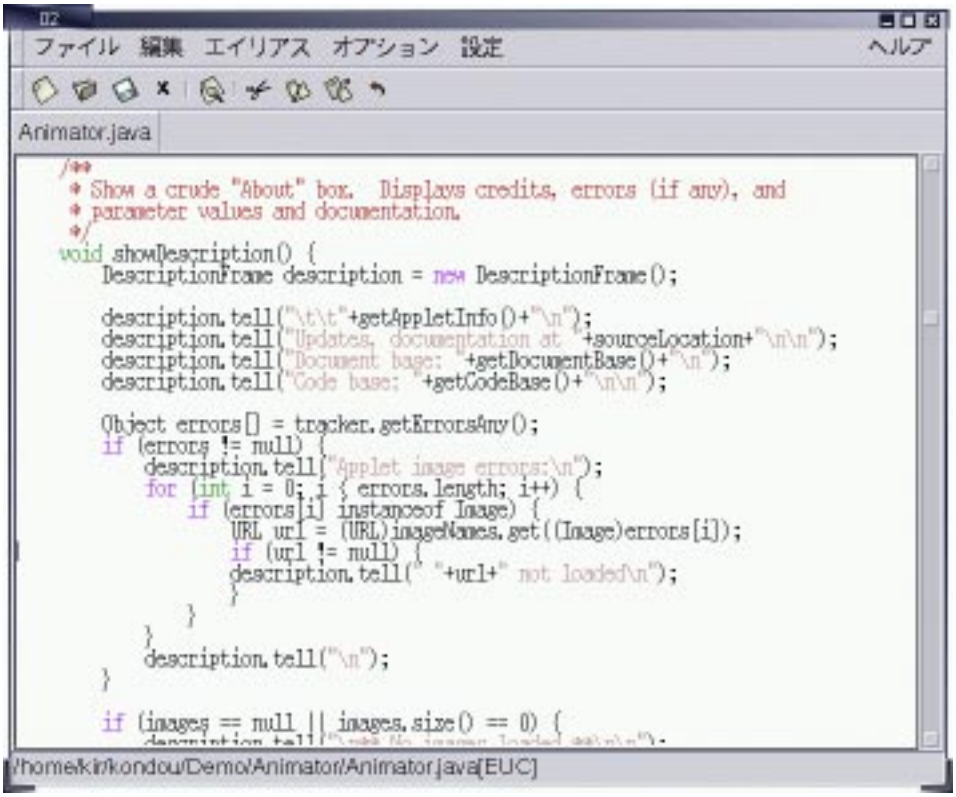


図 12: 通常時のメインウィンドウ

5.2 エイリアス解析の実行

プログラムを起動させると，図 12 のような画面になる．この状態は，まだエイリアス解析を行なっていない状態であり，メニューバーで

「エイリアス」⇒「エイリアスフローグラフ構築」⇒「全体を解析」

を選択する，または「Alt+C」を押すことでエイリアス解析を実行できる．解析中はエイリアスフローグラフ構築中ウインドウが表示され，解析の進捗状況が容易に把握できる．

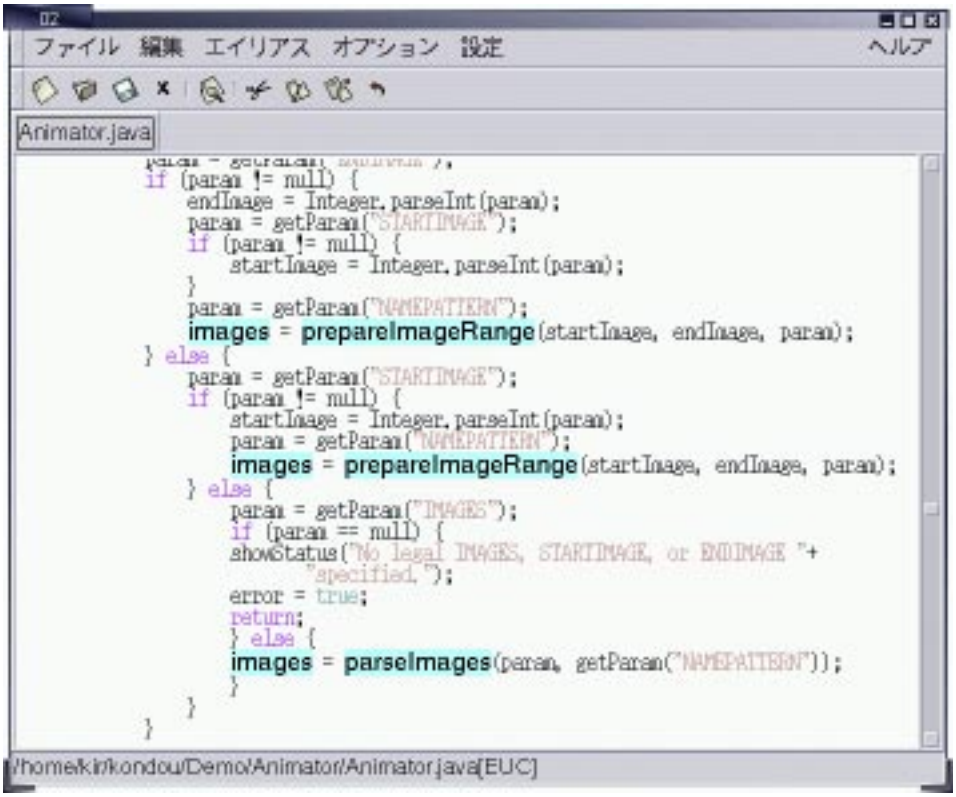


図 13: エイリアスの表示

5.3 エイリアスの表示

エイリアス解析が終了すると，いつでもエイリアスが抽出可能な状態になる．エイリアスの抽出と表示は，ソースコード中のエイリアス指定としたい部分を，マウスのセンターボタンでクリックしてポップアップメニューを表示させ，

「エイリアス表示」

を選択する，またはカーソルが対象上にある状態で「Alt+A」を押すことで行なうことができる．エイリアス表示を行なうと，ソースコード中のエイリアス部分は強調表示され，非エイリアス部分はオプションで指定された表示方法で表示される．初期設定では，非エイリアス部分の表示方法は「保存」となっているので，ソースコードがそのまま表示される (図 13)．

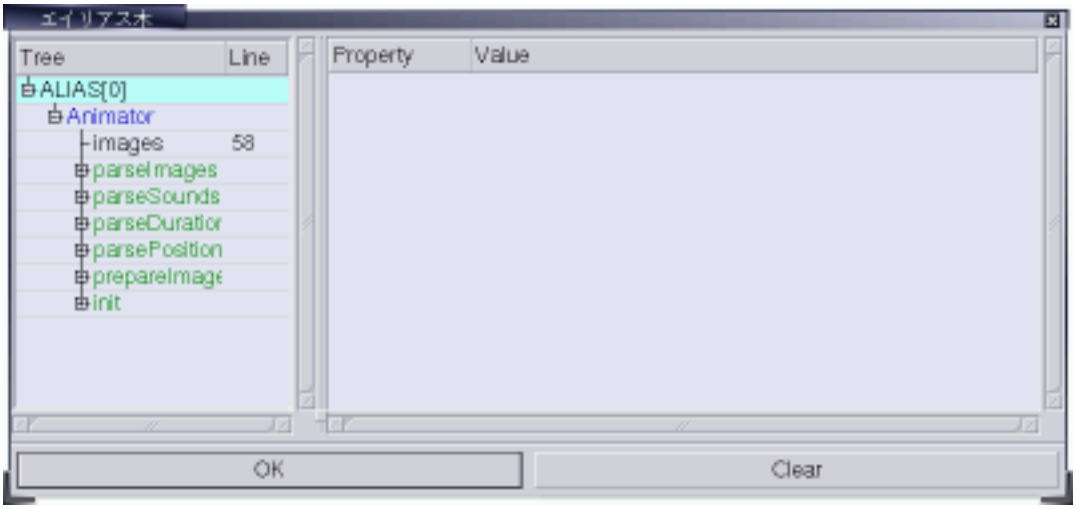


図 14: エイリアスツリーウインドウ

5.4 エイリアスツリーの表示

エイリアスの全体像を把握したいときはエイリアスツリーを用いる．エイリアスツリーの表示は，メニューバーで

「エイリアス」⇒「エイリアスツリー表示」

を選択する，または「Alt+T」を押すことで行なうことができる．表示されたエイリアスツリーウインドウ (図 14) 上のエイリアスツリー (左) を見ることによって，エイリアスがどのクラスのどのメソッド中に分布しているかということを把握することができる．ツリーを作成して表示する時，ツリーのノードをどこまで自動的に展開して表示するかを設定することもできる．これは，メニューバーの

「オプション」⇒「エイリアスツリー表示」

で,「クラスとメソッド」,「クラスのみ」,「全て表示」の三つから選択することで設定できる．初期設定では「クラスとメソッド」になっているので，ツリーを作成した時点では，エイリアスを含むメソッド，クラス名のみが表示され，エイリアスノードはメソッドノードを展開した時に表示される．

エイリアスノードを左クリックすると，右側にそのエイリアスについての情報が表示する (図 15) と同時に，テキストウインドウ上のソースコードで対応する部分にカーソルを移動させ，そのエイリアスをズームアップして表示する (図 16)．

また，エイリアスツリーは，ウインドウを閉じて開きなおすことなく，新たなエイリアスを抽出する度に，そのエイリアスに関するサブツリーを動的に追加する．

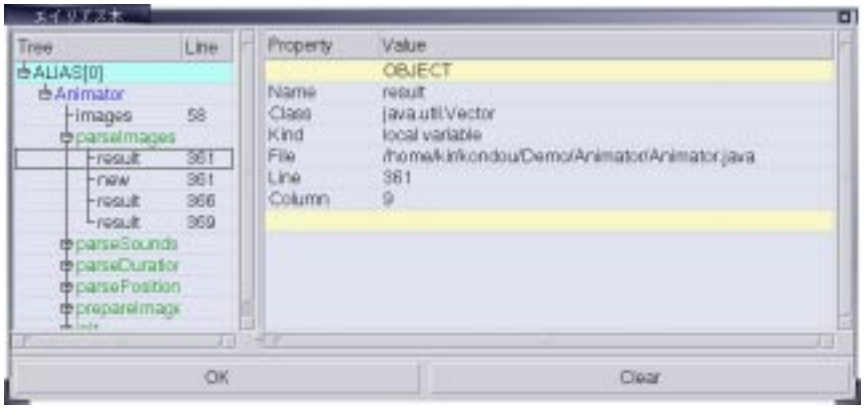


図 15: ノード選択時のエイリアスツリーウインドウ

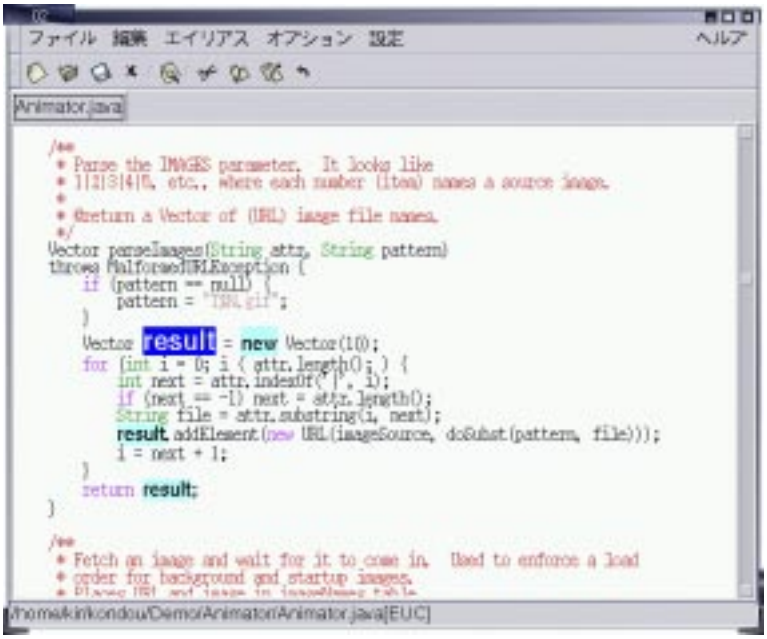


図 16: ノード選択時のメインウィンドウ

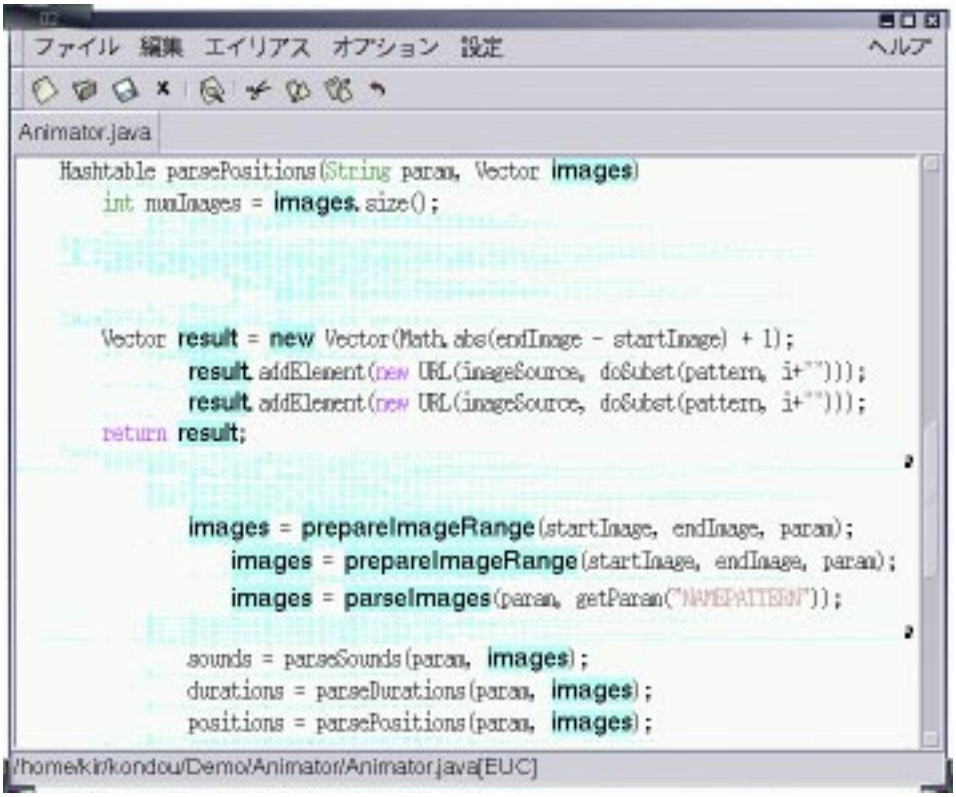


図 17: 非エイリアス表示:圧縮 でのエイリアス表示

5.5 非エイリアス部分表示の変更 1

次に非エイリアス部分の表示方法を変更して，エイリアス表示を行なってみる．非エイリアス部分の表示方法は，メニューバーの，

「オプション」⇒「非エイリアス表示」

で,「保存」,「可変」,「線分化」の三つから選択することができる．ここでは「線分化」を選択する．この状態でエイリアスを抽出して表示すると，エイリアスを含まない行は線で表示される (図 17)．スクロールバーの長さを図 12 と比較しても確認できるるように，1 行が線になるためソースのサイズが大幅に小さくなり，ソースコード 中に分布したエイリアスの確認が容易になる．

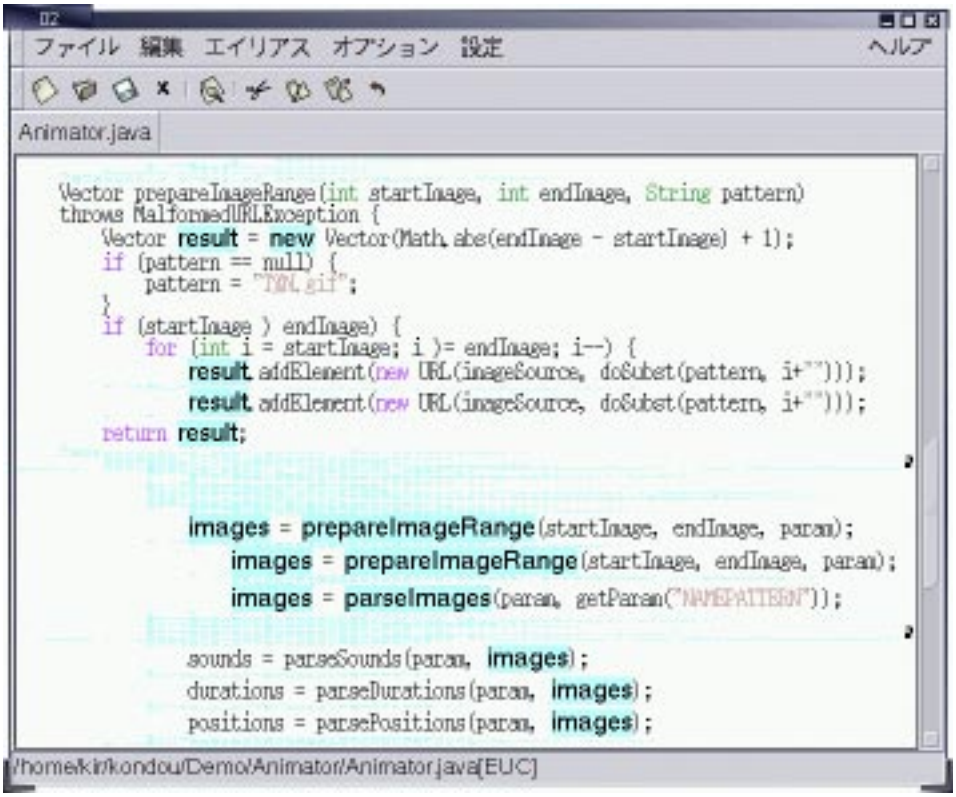


図 18: 視界の拡大

5.6 視界の拡大

非エイリアス部分の表示を「線分化」にしたので，エイリアスを含む行以外のコードは表示されていない．ここで，視界の拡大を行なってみる．視界を拡大したい (付近を見たい) エイリアスを左クリックしてカーソルを移動させ，メニューバーの，

「エイリアス」⇒「スコープ設定」

で，見たい方向を選択できる．実際にはメニューバーから選択するより，ショートカット (前方:Alt+1, 後方:Alt+2, 前後:Alt+3) を利用する方が連続動作が可能となるの現実的である．これを実際に使用して，図 17 の

```
IVector result = new Vector(Math.abs(endImage - startImage)  1);+
```

の前後に視界を広げてみる．まずエイリアス「result」を左クリックする．あとは,「Alt+1」を一回押すごとに一行上へ,「Alt+2」を一回押すごとに一行下へ,「Alt+3」を一回押すごとに一行下へ，視界が広がる．ここでは,「Alt+1」を 2 回,「Alt+2」を 5 回押し，メソッド全体に視界を広げてみた (図 18)．図 17 と比較すると広がっている部分がよくわかる．

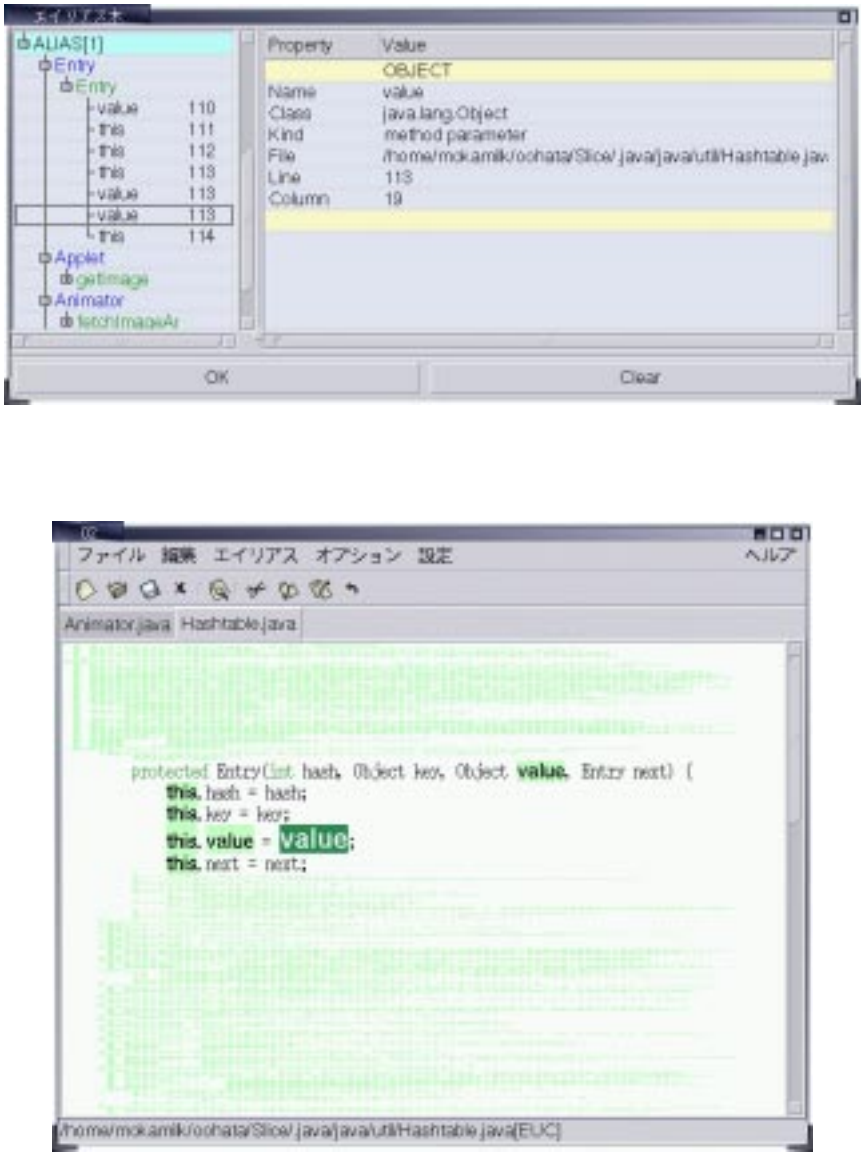
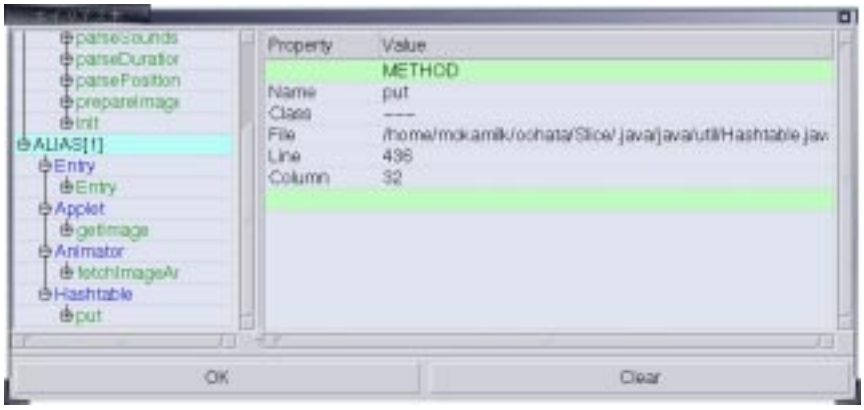


図 19: エイリアスツリーからのファイルオープン (エイリアス選択時)

5.7 エイリアスツリーからのファイルオープン

現在オープンされていないファイルに存在するエイリアスノードを選択した場合も，メニューからファイルをオープンすることなく，自動的にファイルをオープンし，そのエイリアスをズームアップして表示する．そのファイルに存在する他のエイリアスも，自動的に強調表示されている（図 19）．エイリアスノードではなく，クラスノード，メソッドノードを選択した場合でも，同様の処理を行なうことができる (図 20).



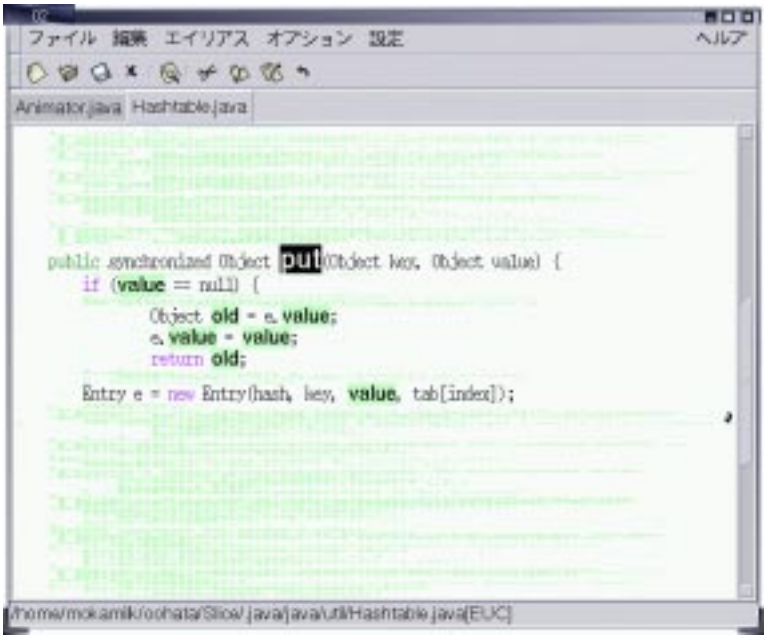


図 20: エイリアスツリーからのファイルオープン (メソッド選択時)

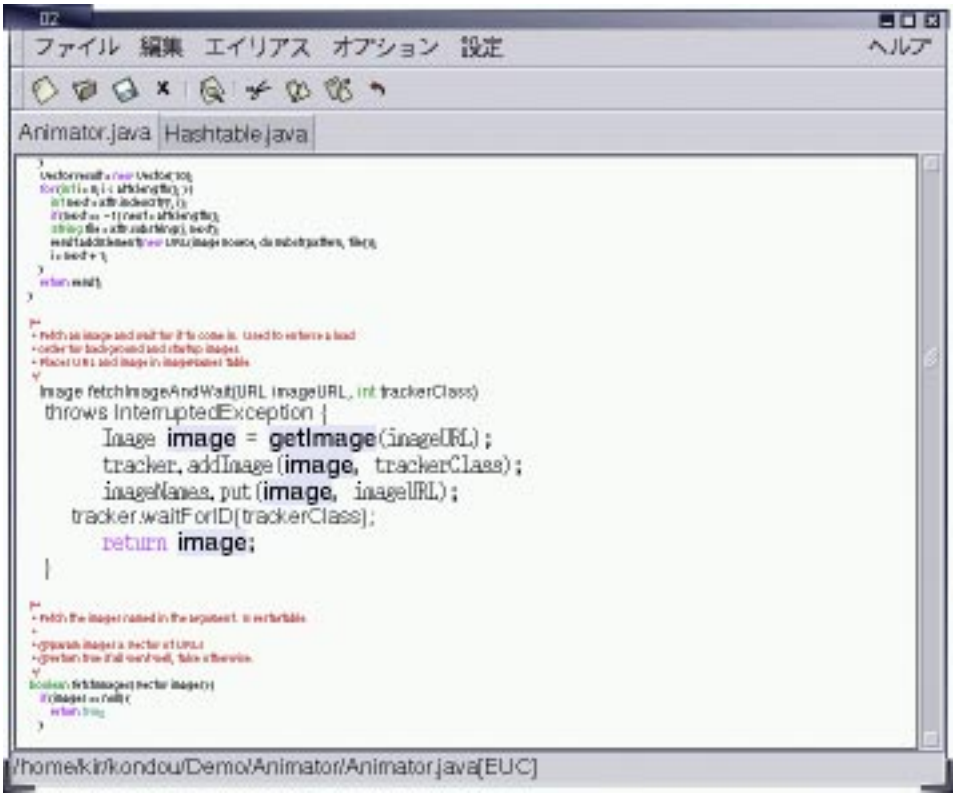


図 21: 縮小表示

5.8 非エイリアス部分表示の変更 2

非エイリアス部分の「可変」表示も「線分化」と同様に行える (図 21). メニューバーの

「オプション」⇒「非エイリアス表示」

で,「可変」を選択すればよい. この状態での視野の拡大も全く同様に行うことができる.

5.9 エイリアスツリーの整理

エイリアスツリーの整理を行ないたい場合はエイリアスツリーウインドウの「Clear」ボタンをクリックすればよい. エイリアスツリーを整理すると図 22 のようになる. 整理された状態でも, ノードをクリックするとエイリアス指定の情報が表示される.

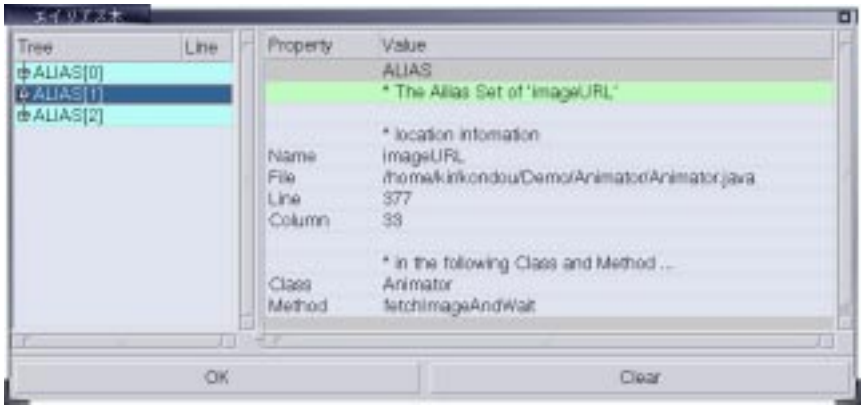


図 22: エイリアスツリーの整理

5.10 オブジェクト情報ダイアログの表示

オブジェクト情報ダイアログは，情報を見たいオブジェクト上でマウスのセンターボタンをクリックし，ポップアップメニューから，

「オブジェクト情報」

を選択することで，表示できる（図 23）．表示された各情報の右にある「移動」ボタンをクリックするとソース上の対応する個所に瞬時に移動することができる．

図 23: オブジェクト情報ダイアログ

6　ツールの使用例と有効性

本研究はエイリアス表示ツールの作成，つまり GUI の作成であるため，ツールの数値的な評価は行い難い．したがって，本ツールを用いたデバッグ例を挙げることでツールの有効性を示す．

JDK 付属のサンプルコードに SpreadSheet.java(サイズ: 約 1000 行) というプログラムがある．これは簡単な表計算を行うJAVA アプレットである．このソースコードを編集しているうちに，図 24 のようなエラーが発生した．

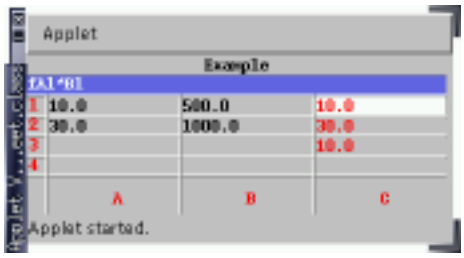


図 24: 表計算アプレット SpreadSheet のエラー例

表中の C1(選択部) は $fA1 * B1$，つまり A1 と B1 の積が表示されるべきである．ところが，ここでは 10 が表示されており，積になっていない．そこで，この計算式の解析を扱うメソッド parseFormula の引数である String 型のオブジェクト formula についてのエイリアスを表示させ，デバッグを行うことにした．

エイリアスツリー (図 25) より，エイリアスはクラス Cell 中のメソッド parseFormula と parseValue 中にのみ存在することが把握できた．また，ソースコード中の非エイリアス部分の表示方法は「線分化」を指定したため，ソースコード中に含まれる全てのエイリアスをも把握することができた (図 26)．そこで，エイリアスツリーウインドウの情報表示リストにエイリアス情報を表示させながら，エイリアスを含む文のみを調べていくと，メソッド parseValue の最後の戻り値が formula となっていた．これは情報表示リストでも確認すると

```
Kind      method parameter
```

となっており (図 27) 仮引数をそのまま返していたことになる．よって，この付近のソースを表示させ集中的に調べたところ正しくは restFormula を返さなければならないことに気づき，`return restFormula` に変更して再度実行を行ったところ，図 29 のように正しい実行結果が得られた．

このようにエイリアス表示を用いることで，1000 行ほどのプログラムでも箇所をしぼり込んでデバッグを行うことがでる．またエイリアスツリーウインドウを用いることで，エイリアス全体像を把握しつつ，エイリアス一つ一つの詳細な情報を表示させることが可能とな

り，デバッグ，プログラム理解などに有用であると思われる．

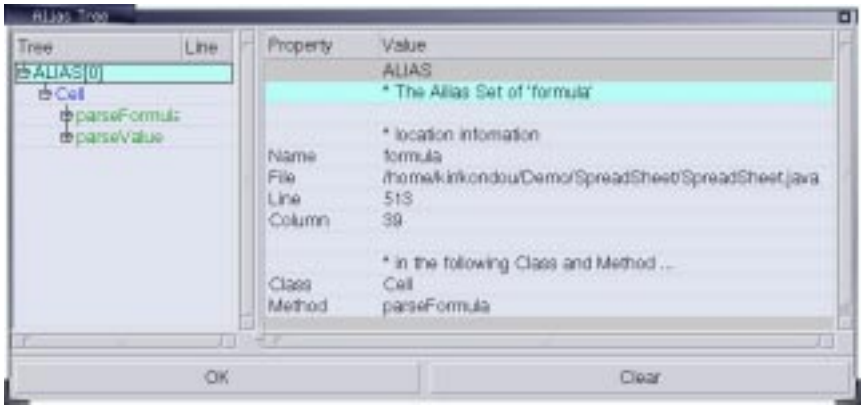


図 25: エイリアス表示直後のエイリアスツリーウインドウ

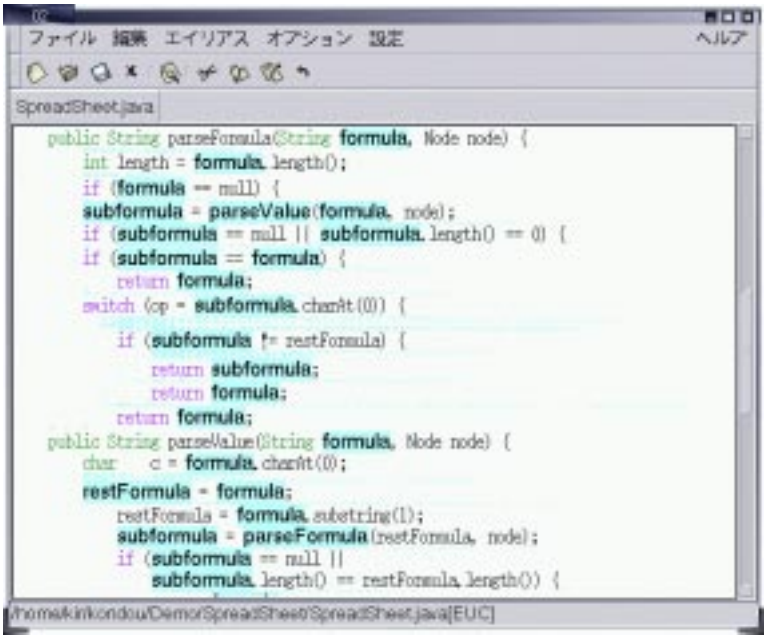


図 26: エイリアス表示直後のメインウィンドウ

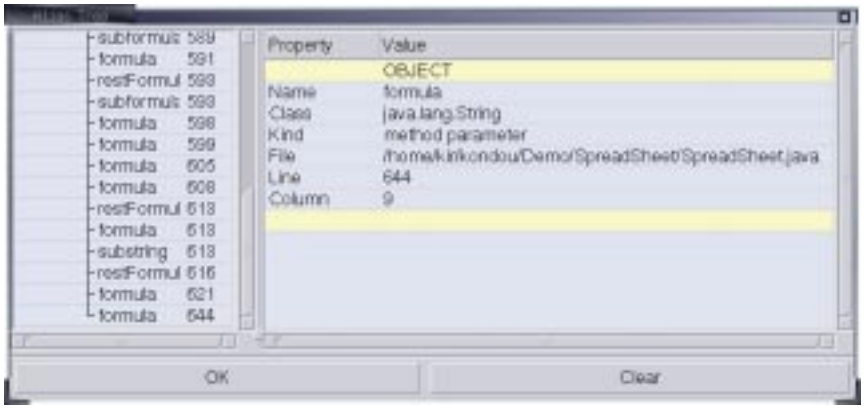


図 27: エイリアス formula の情報を表示させたエイリアスツリーウインドウ

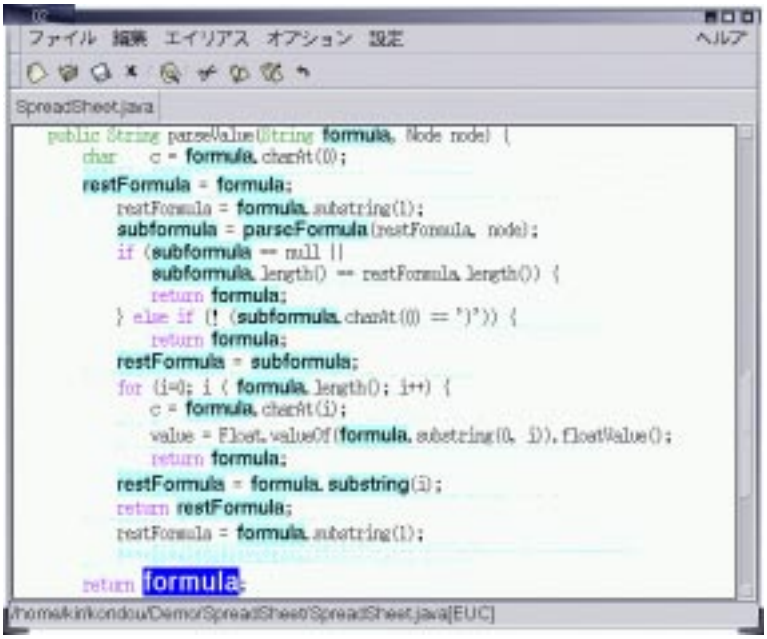


図 28: エイリアス formula に視点を置いたメインウィンドウ

図 29: 表計算アプレット SpreadSheet の正常な実行例

7 まとめと今後の課題

本研究では，解析部から渡された Java プログラムのエイリアス解析結果を利用して，エイリアス集合の各要素ごとの詳細な情報をソースコードと対応させながら表示でき，エイリアス情報を有効に利用するための様々な機能を有するツールの試作を行なった．大きな機能として，

- テキストウインドウ上での強調表示：縮小 (可変，線分化)，保存
- エイリアスツリーウインドウによるエイリアス全体像の把握
- テキストウインドウとエイリアスツリーウインドウの連係処理

を設計・実装した．

今後の課題としては，

- スライス表示ツールへの発展
- ツールの有効性の評価
- 詳細な制御構造の理解を可能とする非エイリアス部分の表示
- 3D 利用によるエイリアス部分と非エイリアス部分の区別

などが挙げられる．

謝辞

本論文の作成において，常に適切な御指導および御助言を頂きました 大阪大学 大学院基礎工学研究科 情報数理系専攻 ソフトウェア科学分野 井上 克郎 教授に深く感謝致します.

本論文の作成において，常に適切な御指導を頂きました 同 楠本 真二 助教授に深く感謝致します.

本論文の作成において，常に適切な御助言を頂きました 同 松下 誠 助手に深く感謝致します.

本論文の作成において，常に適切な御助言を頂きました 同 大畑 文明 氏に深く感謝致します.

最後に，その他の面で様々な御指導，御助言を頂いた 大阪大学 大学院基礎工学研究科 情報数理系専攻 ソフトウェア科学分野 井上研究室の皆様に深く感謝致します.

参考文献

[1] T. Ball, and S. G. Eick, “Visualizing Program Slices,” in Proceedings of the 1994 IEEE Symposium on Visual Languages, pp.288–295, 1994.

[2] M. H. Brown,“ Zeus: A system for algorithm animation and multi-view editing,” In 1991 IEEE Workshop on Visual Languages, pp.4–9, 1991.

[3] S. K. Card, G. G. Robertson, and J. D. Mackinlay, “The Information Visualizer, an information workspace,” Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI’91)

[4] J. Gosling, B. Joy, and G. Steele, 村上 雅章 [訳], “The Java 言語仕様.”

[5] M. Hind, and A. Pioli, “An Empritical Comparison of Interprocedural Pointer Alias Analysis,” in IBM Research Report #21058, 1997.

[6] J. D. Mackinlay, G. G. Robertson, and S. K. Card, “The perspective wall: detail and context smoothly integrated,” In Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI’91), pp.173–179. ACM Press, 1991.

[7] D. R. Musser, and A. Saini, 滝沢 徹, 牧野祐子 [訳], “STL 標準テンプレートライブラリによる C++プログラミング,” Addison-Wesley.

[8] S. P. Reiss, “An Engine for the 3D Visualization of Program Information,” Department of Computer Science Brown University, 1995.

[9] J. L. Steffen, S. G. Eick, and E. E. Sumner Jr., “ Seesoft - a tool for visualizing line-oriented software statistics,” IEEE Trans. on Software Engineering, Vol. 18, No. 11, pp.957–968, 1992.

[10] B. Stroustrup, “The C++ Programming Language(Third edition),” Addison-Wesley, 1997.

[11] M. Weiser, “Program Slicing,” in Proceedings of the 5th International Conference on Software Engineering, pp.439–449, 1981.

[12] 大畑 文明, “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の提案と実現,” 大阪大学基礎工学研究科修士学位論文, 2000.

[13] 小池 英樹, 石井威望, “フラクタルの概念に基づく提示情報量制御手法,” 情報処理学会論文誌, Vol. 33, No. 2, pp.101–109, 1992.

[14] 塩澤 秀和, “インターネットの視覚化「納豆ビュー」,” 日経サイエンス別冊 125 変わるネット社会, pp.86–89, 1998.

[15] 竹田 英二, “GTK+ではじめる X プログラミング,” 技術評論社.

[16] 田中 ひろゆき, “GTK+入門 基礎からはじめる X プログラミング,” ソフトバンク.

[17] 日本サン・マイクロシステムズ株式会社，サン・サービス エデュケーションサービス部，“Java プログラミング講座，” アスキー出版局.

[18] 平川 正人, “マルチメディアソフトウェア工学,” 共立出版.

[19] 増井 俊之，“情報視覚化の最近の研究動向,” 電子情報通信学会第 9 回データ工学ワークショップ, 1998.

[20] `http://lazy.ton.tut.fi/terop/iki/gtk/gtk--.html`, “Gtk−−.”

[21] `http://www.gtk.org/`, “GTK+ − The GIMP Toolkit.”