意味に基づいたJavaプログラムの分類手法

渡辺　義大　　武田　正之

東京理科大学大学院　理工学研究科　情報科学専攻

1．はじめに

プログラムが類似なものかを検証することは,類似部分のマクロ化[1],盗用の発見,アルゴリズムの判定等に利用できる.

プログラム中の類似部分のマクロ化を行うツールとして,Clone-DRがある.Clone-DRは,プログラム中の類似な部分（クローン）を発見して,クローンをマクロに置き換える.その方法とは,プログラムを構文解析し,生成された木構造におけるノード同士の比較を行い,類似な部分を発見したら,マクロへの変換を行うやり方である.しかし,プログラムの意味が重視されていなかった.

本研究では,プログラムの意味に基づいた分類手法について提案する.本手法では,Java プログラムに対して,変数の依存関係,変数のデータ構造等,プログラムの意味を表す部分に着目し,分類を行っている.これにより,アルゴリズムに基づいたマクロ化が可能となる.

プログラムが盗用されたものであるかの判定は,従来,プログラム中に透かし情報を埋め込むことにより行われていた.透かし情報とは,プログラムの実行結果には影響しないよう,変数名や式を変形させたビット情報である.この透かし情報を検出することにより,プログラムの著作権を保護することができる.しかし,この方法では,透かし情報を埋め込む場所,埋め込む方法等が解析されてしまった場合には,効果を発揮できない.

本研究では,プログラムの意味,特に変数の依存関係に着目して判定を行うため,アルゴリズムの同等性を検証でき,盗用の有無の検証へ発展させることも可能である.さらに,複数のプログラムに対して判定することにより,アルゴリズムによる自動分類も行える可能性がある.

本稿では,Java プログラムの分類のための**類似性判定レベル**について定義し,さらに分類を行うシステムの実装方法について述べる.

2.Javaプログラムの分類手法

Java プログラムの分類は,プログラム構成要素の類似性により行うが,以下ではその判定基準を述べる.

Semantic Based Clustering System for Java Program
by Yoshihiro Watanabe and Masayuki Takeda (Graduate School of Science and Engineering Tokyo University of Science)

2.1. 類似性判定の基準

原則として,プログラムを構文解析した木構造をトップダウン的に辿ることにより,類似性の判定を行っていく.しかし,それだけではプログラムの意味をとらえることができない.そのため,木構造を辿る深さとは別に,以下に示すレベルを用いて判定を行う.

レベル1：クラスの修飾子,継承元

クラスの修飾子が,public であるか private であるかということは,プログラムが他のクラスに対する役割を意味する.また,クラスの継承元を調べることは,クラスの性質,役割を表すものであり,重視する必要がある.

レベル2：変数の依存関係

依存関係とは,変数の値がメソッド内で変更されているか,また,変数がメソッドの返り値に影響を与えているか等の関係である.この依存関係は,プログラムを特徴付ける意味として重要なものであり,依存グラフにより表現する.

レベル3：変数のデータ構造

クラス内のインスタンス変数が,特徴的なデータ構造を持っている場合,それによりプログラムの特徴を見ることができる.

レベル4：メソッドのシグニチャ

メソッドのシグニチャ（型,引数等）は,メソッドの特徴の一つである.メソッドは,定義する順番が前後していてもかまわない.メソッド名は無視して判定するが,メソッドの型,引数の個数,引数の並び順が違った場合は,別のメソッドとして考える.

レベル5：メソッドの内部構造

メソッドの内部にどのような構造があるかを調べることにより,メソッドを特徴づけることが可能となる.内部構造としては,ループ文（while,for 文）,条件分岐（if 文,switch 文）がある.ループ文に関しては,while が for で表現された場合でも、同等のものと判断する.

2.2.実装方法

Jikes[2]を用いて Java プログラムを構文解析し,抽象構文木 JavaML[3]を生成する. JavaML は,Java ソースプログラムを XML 形式に変換したものであり,この上で類似性判定を行う（**図1**）.

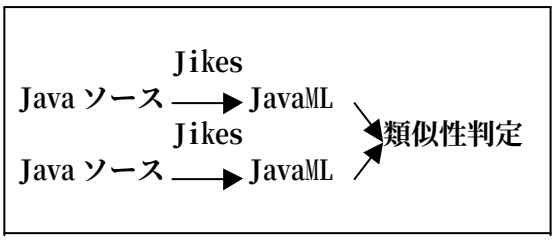

図１　類似性判定の流れ

3.分類について

3.1.バイトコードに対する分類

Java の場合,バイトコードに対して分類を行う方法が考えられる.しかし,バイトコード上では,Java コンパイラの最適化処理により,局所変数がスタックフィールドに置き換えられてしまう可能性がある.その場合,変数の依存関係を追えなくなり,バイトコードに対する分類では,詳細な依存関係を調べることが困難となる.

3.2.類似性の例

ここでは,レベル 5 で行うループ文に対しての類似性判定について,例を挙げる.

＜例１＞

```
int count = 0;
for(int i = 0; i < 5; i++){count++;}
```

＜例２＞

```
int i = 0,count = 0;
while(i < 5){count++; i++;}
```

例１は,for 文によるループであり,例２は,while 文のループである.この２つのループで,内部が実行される回数は 5 回である.よって,count の値は結果として両方とも 5 になる.これより ,例 1 と例２のループは,同じ意味となるので,類似なもの同士と判定できる.このように,繰り返し条件の判断は,静的に判定できる範囲で行う.

3.3.依存関係に基づいた分類の例

依存関係に基づく分類の例を次に示す.

<例３>

```
class A {
  int s;
  int addA(int i){
   s = i + s;
   return s;
 }
}
```

```
class B {
  int t;
  int addB(int j){
    j = t + j;
    return j;
  }
 }
```

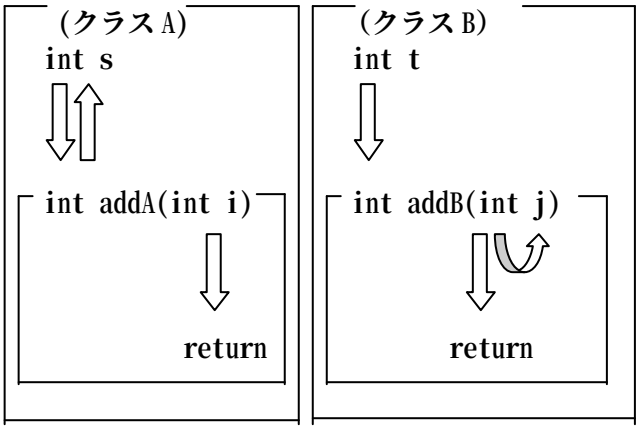

図２　例３の依存グラフ

図２は,例３のプログラムの依存グラフである.変数からメソッドに向けて出されている矢印は,変数がメソッド内で参照されていることを表す.また,メソッドから変数に向けて出されている矢印は,メソッド内で,その変数の値が変更されたことを表す.

クラス A には、int 型のインスタンス変数 s が存在している.変数 s は addA メソッド内で引数 i だけ加えられる.よって addA メソッドを実行すると,s の値が変更されることが分かる.

一方,クラス B には,int 型の変数 t が存在している.addB メソッドを実行すると,引数 j と t の和 j を返す.よって,addB メソッドを実行しても t の値は変更されない.

構文解析して,変数とメソッドのシグニチャ同士を比較しても,クラス A とクラス B の違いを判別することはできない.しかし,レベル２で用いている依存関係を考えると,これらは類似でないクラスと判定できる.

4.おわりに

本稿では,Java プログラムの分類を行う際に,意味に基づいた手法を提案した.本手法を用いれば,変数の依存関係等を考慮した分類を行うことができる.今後の課題として,マクロ化,盗用防止,アルゴリズム判定に有用であるかの検証を行いたい.

参考文献

[1]井上克郎,神谷年洋,楠本真二,”コードクローン検出法”,コンピュータソフトウェア,Vol,18,No5,pp.47-54(2001)
[2]Jikes,http://oss.software.ibm.com/developerworks/opensource/jikes/
[3]JavaML,http://www.cs.washington.edu/homes/gjb/JavaML/