5L-7

# 仮想メソッド呼出しを用いることによるプログラム難読化のオーバヘッド削減

石田峰文 †　　小宮常康 †
電気通信大学大学院情報システム学研究科 †

## 概要

プログラムの制御フローの難読化として、条件分岐を用いて実際には実行されないコードや崩れた制御構造を作り出す手法が存在するが、条件分岐によるオーバヘッドを伴う。そこで、難読化で加える条件分岐の動作を仮想メソッド呼出しで実現する手法を提案する。この仮想メソッド呼出しにおける多態性を実現するメソッドディスパッチが、条件分岐と同等の役割を果たす。しかし、この仮想メソッド呼出しの実行時の振舞いは多態的ではなく、動的最適化によるインライン展開によってメソッドディスパッチの除去が可能である。それにより、難読化で加わるオーバヘッドの削減を期待できる。

## 1　背景

プログラムのアルゴリズムが静的に解析されることを困難にするため、プログラムを実行したときに通る制御の流れ (制御フロー) を複雑なものに変換する手法が使われることがある。これを制御フローの難読化と呼ぶ。Collberg ら [1] は、Opaque Predicate を用いた制御フローの難読化として、デッドコードの挿入や制御構造パターンの破壊などについて述べている。Opaque Predicate は、難読化を行う時点で取りうる値が決まっている述語である。例として、整数変数 $b := 6$ に対する条件式 $b > 5$、任意の整数変数 $x, y$ に対する条件式 $7x^2 - 1 \neq y$ は、真という値を取ることが決まっている Opaque Predicate である。

制御フローの難読化の欠点として、制御フローを複雑にすることにより難読化されたプログラムにオーバヘッドが加わることがある。先の Opaque Predicate を用いた難読化では、条件分岐の実行によるオーバヘッドが増加する。

プログラムを高速に実行するため、動的最適化を行い高速なコードへ変換する手法が使われることがある。これを実行時コンパイル (JIT コンパイル) と呼ぶ。Kotzmann ら [2] は、Java の JIT コンパイラで一番影響の大きい動的最適化は仮想メソッド呼出しのインライン展開であるとまとめている。仮想メソッド呼出しには、プログラムに表記された静的な型ではなく実行時の型から呼出されるメソッドが決まる、多態性と呼ばれる性質がある。しかし実際に呼出すメソッドが複数あることは少ない。そのため、実行の際に頻繁に呼出されるメソッドに対してインライン展開を行い、展開されたコードが実行すべきコードかどうか直前に調べ (ガードと呼ぶ)、問題が無いならば展開されたコードを実行する [3]。

## 2　提案手法

本研究では、Opaque Predicate として条件分岐の代わりに仮想メソッド呼出しを用いる手法を提案する。

仮想メソッド呼出しの多態性は、オブジェクトの型を条件として呼出すメソッドを決めると見なすことができる。そのため、条件式の値とオブジェクトの型を対応させることで、仮想メソッド呼出しを条件分岐と同等の働きをさせることができる。

図 1 に、文 $\mathtt{s_i}$ に対し Opaque Predicate を用いたデッドコード挿入による難読化を行った際の制御フローグラフを示す。実線のフローが実際に実行されるフロー、背景色のあるノードは難読化により加わったオーバヘッドとなるノード、点線のフローとノードは実行されないダミーのノードとフローである。Opaque Predicate として、図の左が条件分岐を用いた既存手法、図の右が仮想メソッド呼出しを用いた提案手法である。ここで、$\mathtt{P^T(x,y)}$ は引数 $x, y$ を取る恒真な条件式、$\mathtt{b[x][y]}$ は $\mathtt{P^T(x,y)}$ 結果からなるブール型の配列、$\mathtt{o[x][y]}$ は $\mathtt{P^T(x,y)}$ の結果から決まるオブジェクトの配列、$\mathtt{o^A}$ はクラス A を、$\mathtt{o^B}$ はクラス B を型とするオブジェクトである。図 1 から、仮想メソッド呼出しは条件分岐とほぼ同じフローにすることができることが言える。

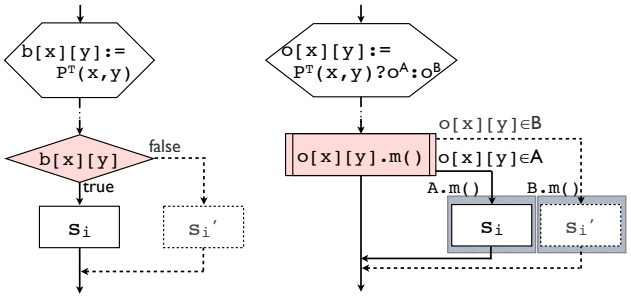


図 1　$\mathtt{s_i}$ に対するデッドコード挿入による難読化

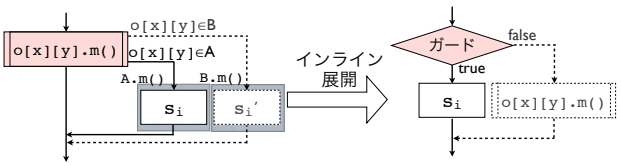


図 2　仮想メソッド呼出しのインライン展開後の制御フロー

Reducing the overhead of obfuscated programs by using virtual method invocations.
†Takafumi ISHIDA and Tsuneyasu KOMIYA, The University of Electro-Communications.

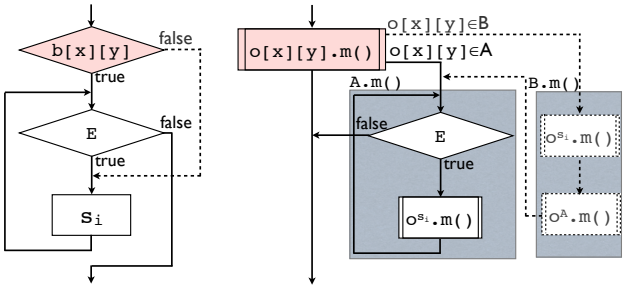


図 3　制御構造パターンの破壊による難読化の例

仮想メソッド呼出しはオーバヘッドの大きい処理であるため、このままでは実行速度は遅くなってしまう。しかし、実行時の最適化処理により仮想メソッド呼出しのインライン展開が行われることで、速度低下を抑えることができる。特に、本手法では仮想メソッド呼出し時のオブジェクトの型を 1 つに固定しているため、インライン展開の効果が最大限に活かされる。図 2 に図 1 の仮想メソッド呼出しがインライン展開されたときの制御フローグラフを示す。

以上から、難読化で用いる恒真恒偽の条件式を仮想メソッド呼出しで代用しても、インライン展開後のオーバヘッドはほぼ等しくなると考えられる。更に、仮想メソッド呼出しの最適化は JIT コンパイラにとっても重要であるため [2]、より高度な最適化が行われることでオーバヘッドが削減される可能性も見込まれる。

本手法は Opaque Predicate である条件分岐を仮想メソッド呼出しで代用するため、Opaque Predicate を用いた難読化の全てに適用することができる。図 3 に、制御構造パターンの破壊による難読化を行った際の制御フローグラフを示す。左が条件分岐と goto を用いた既存手法、右が仮想メソッド呼出しを用いて同様の制御フローを表現した提案手法である。

## 3　評価

以下の Java プログラムを用いて、難読化に伴うオーバヘッドを既存手法と提案手法において測定し比較した。test1 メソッドは難読化される前のコードであり、その 14 行目を難読化したコードが test2(既存手法) と test3(提案手法) である。既存手法では、22 行目の条件分岐は真偽が静的な解析では分からないため、どちらが実行されるか分からないため難読である。提案手法では、30 行目の仮想メソッド呼出しはクラス A とクラス B のどちらのメソッド m を呼出すか静的な解析では分からないため、33 行目と 36 行目のどちらが実行されるか分からないため難読である。

```
1  //初期化
2  for(int x=0;x<100;x++){
3      for(int y=0;y<100;y++){
4          b[x][y]=Pt(x,y);
5          o[x][y]=Pt(x,y)?O.a:O.b;
6      }
7  }
8
9  /** オリジナル */
10 static void test1(){
11     for(int i=0;i<cnt;i++)
12         for(int j=0;j<100;j++)
13             for(int k=0;k<100;k++)
14                 val++;
15 }
16
```

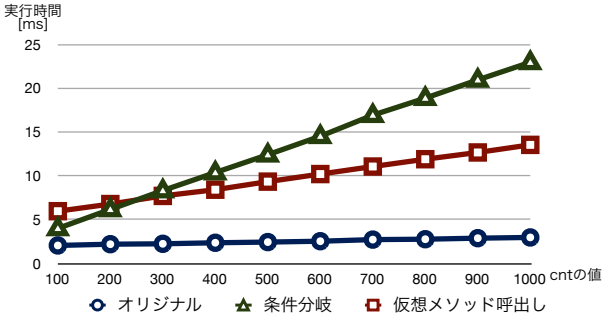


図 4　メソッドの平均実行時間 (20 回実行)

```
17 /** 条件分岐を用いた既存手法 */
18 static void test2(){
19     for(int i=0;i<cnt;i++)
20         for(int j=0;j<100;j++)
21             for(int k=0;k<100;k++)
22                 if(b[j][k]) val++; else val--;
23 }
24
25 /** 仮想メソッド呼出しを用いた提案手法 */
26 static void test3(){
27     for(int i=0;i<cnt;i++)
28         for(int j=0;j<100;j++)
29             for(int k=0;k<100;k++)
30                 o[j][k].m(i,j,k);
31 }
32 class A extends O{
33     void m(int i,int j,int k){val++;}
34 }
35 class B extends O{
36     void m(int i,int j,int k){val--;}
37 }
```

Java の仮想マシンとして、Oracle の HotSpot のサーバ版仮想マシン (1.7.0_04-b21,build 23.0-b21) を用いた。プログラム中の cnt の値を 100〜1,000 で変化させたときのメソッドの平均実行時間のグラフを図 4 に示す。図 4 から、条件分岐よりも仮想メソッド呼出しの方がオーバヘッドが低くなることが実際にあることを確かめることができた。

## 4　結論と今後の課題

本研究では、プログラムの制御フローの難読化において加えられた条件分岐に対し、同等の動作をする仮想メソッド呼出しを用いる手法を提案した。実際に Java のプログラムで実行時間の評価を行ったところ、既存手法よりオーバヘッドが減らすことができることがあることが判明した。今後は、提案手法を自動的に施す変換器を実装し、より複雑なプログラムを対象に様々な観点から提案手法の評価を行う。

## 参考文献

[1] Christian Collberg, Clark Thomborson and Douglas Low: “Manufacturing Cheap, Resilient, and Stealthy Opaque Constructs”, *Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pp. 184–196, 1998.

[2] Thomas Kotzmann, Christian Wimmer, Hanspeter Mössenböck, Thomas Rodriguez, Kenneth Russell and David Cox: “Design of the Java HotSpot™ Client Compiler for Java 6”, *ACM Transactions on Architecture and Code Optimization*, Vol. 5, Article 7, 2008.

[3] David Detlefs and Ole Agesen: “Inlining of Virtual Methods”, *ECOOP ’99*, pp. 258–277, 1999.