B-026

Java プログラムの逆方向解析
Reverse Analysis of Java Programs

稲船 哲也† 平中 幸雄† 武田 利浩†
Tetsuya Inafune Yukio Hiranaka Toshihiro Taketa

1. はじめに

現在、プログラムのテストに使われているツールは順方向解析を利用しているものが多い。順方向解析とは、プログラムが入力に対して出力した結果が想定内の出力になっているかどうかでバグの有無を判断する方法である。しかし、順方向解析の欠点はテストを行った入力パターンの、バグの有無しか確認できないことである。これでは、プログラムに安全性の問題が残っている可能性がありながら運用していくことになる。

そこで、本研究ではプログラムに安全性の問題が残っているかどうかを明らかにするために、出力から逆方向にプログラムを解析することでその原因を求めるテストプログラムの開発を行う。

2. 逆方向解析プログラム

2.1 概要

今回提案した逆方向解析は、順方向解析とは逆に入力の情報を利用せず、想定外の出力の情報をプログラムに逆方向に流すことによって、その入力条件があるか調べるものになる。想定外出力の条件となる入力の条件を求めることで、プログラムの安全性を範囲的に検証することができるはずである。図 1 は順方向、逆方向解析の概要図である。

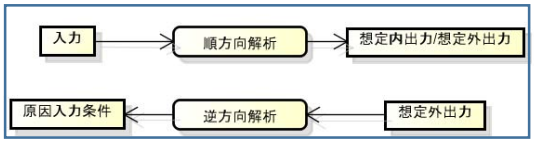

図 1. 順方向、逆方向解析概要図

2.2 解析対象プログラム

本研究では解析対象を、Java の機械語プログラムとしている。この理由は Java の機械語がほかの言語と比べて、命令の種類が少なく、構造が単純化されているので実行の流れが特定しやすく、本研究で最初に取り扱う対象として適していると考えたためである。また、解析対象プログラムの処理内容は数値計算処理を行っているものとしている。

2.3 逆方向解析のアルゴリズムの検討

逆方向解析は図 2 のようなアルゴリズムで行う。

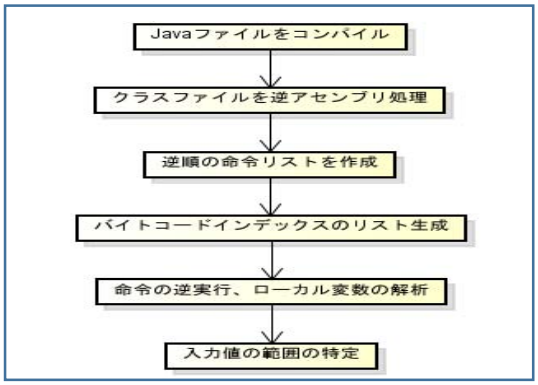

図2. 逆方向解析の方法

まず逆アセンブル処理したクラスファイルから、逆順の実行命令リストを作成する。次に実行命令のバイトコードインデックスを取得する。バイトコードインデックスとは先頭0から始まる命令のバイト番号である。機械語ではこれを利用して条件分岐を行っている。

機械語命令では値をローカル変数とスタックのやりとりによって扱っている。ローカル変数は値を保存できる場所で、スタックは一時的に使う値の置き場所で、逆方向解析を行う上で、ローカル変数に何が保存されているのかを把握する必要がある。そこで命令の逆実行を行いながら、ローカル変数の値の解析を行っていく。ローカル変数の解析のやり方としては、代数的な方法をとり、値が不明な部分は仮の変数を与え、実際の値が分かり次第実際の値と置き換えるというような形で解析していく。具体的には図 4 の解析対象 Test. java に対して、解析結果は図 5 のようになる。

```
public class Test {
        public static void main(String[] args){
                int x=1;
                int y=10;
                int z=x+y;
                }
}
```

図 4. Test. java

		ローカル変数1	ローカル変数2	ローカル変数3
9:return				
8:istore_3				S1
7:iadd				iadd
6:iload_2			L2(0)	+L2(0)
5:iload_1		L1(0)	L2(0)	L1(0)+L2(0)
4:istore_2		L1(0)	S2	L1(0)+S2
2:bipush	10	L1(0)	10	L1(0)+10
1:istore_1		S3	10	S3+10
0:iconst_1		1	10	1+10

図 5. ローカル変数の解析

8 番の命令「istore_3」はスタックの値をローカル変数 3 に保存する命令だが、この時点ではスタックに積まれている値がわからないので仮の変数として 1 つ目の「istore」によって保存された変数として「S1」を置く。

7 番の命令「iadd」はスタックに積まれた 2 つの値を加算した値をスタックに積む命令であり、8 番の命令でローカル変数 3 に保存された値が「iadd」によるものだと判明するので、「S1」を「iadd」という変数に置き換える。

6 番の命令「iload_2」はローカル変数 2 の値をスタックに積むという命令であり、7 番の命令時にスタックに積まれていた片方の値が判明したので、仮の変数として「iload_2」によるローカル変数 2 の値であるのでローカル変数 2 とローカル変数 3 に「L2(0)」を置く。

5 番の命令「iload_1」では 6 番の命令と同じ要領で 7 番の命令で加算されたもう片方の値が判明したので、ここでは仮の変数「L1(0)」を置く。

4 番の命令「istore_2」はスタックの値をローカル変数 2 に保存するという命令なので、6 番の命令でスタックに積まれた値がこの命令によって保存されていたものだったということが判明したので、「L2(0)」を「S2」に置き換える。

† 山形大学 Yamagata University

2 番の命令「bipush 10」は int 型の値 10 をスタックに積むという命令なので、4 番の命令でローカル変数 2 に保存された値が 10 だと判明したので、「S2」を「10」と置き換える。

1 番の命令「istore_1」で「L1(0)」値がここで保存された値だと判明するので、「L1(0)」を「S3」に置き換える。

0 番の命令「iconst_1」はスタックに int 型の数値 1 を積む命令であり、ここで 1 番の命令でローカル変数 1 に保存された値が判明したので「S3」を「1」に置き換える。

このようにしてローカル変数の解析を行う。この情報と条件分岐にから判明する入力の制約条件などを利用して、想定外出力となる入力範囲の特定を行う。

2.4　逆方向解析を行う上での問題と対策

逆方向解析を行おうと考えた場合、いくつか問題がある。1 つ目としては 2 入力 1 出力の演算では、1 出力から 2 入力を一意に決めることができない。演算を逆方向で行おうと考えた場合、すべての入力値の組み合わせを試さなければならなくなる。この問題には、入力に関する制約条件を利用して範囲を限定していくことによって、入力の条件を絞り込んでいくことで対応する。たとえば「x+y=z」の逆演算で入力 x を 0 以上の範囲(0～MAX)と 0 未満の範囲(-MIN～-1)に分割してそれぞれの場合で逆演算を行っていくと「x+y=z」を満たす入力の範囲は図 6 のように限定される。

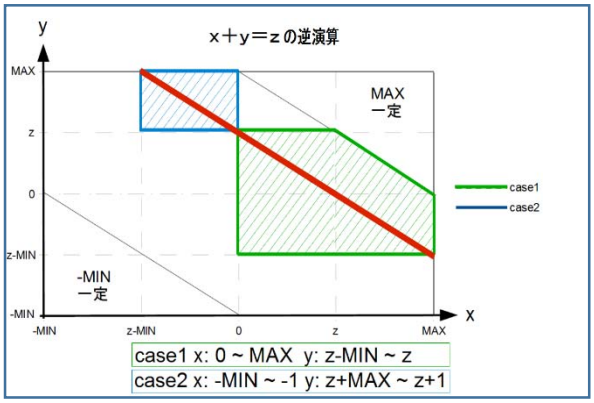

図 6. x+y=z の逆演算

case1 x:0～MAX と case2 x:-MIN～-1 のそれぞれの範囲で「x+y=z」が限定されることになる。ここからさらに機械語命令の条件分岐命令などから得られた入力に関する制約条件から、入力の範囲の分割を増やす、あるいは範囲を狭めることによって、さらに「x+y=z」の逆演算を行い、破綻の可能性がないか確認しながら入力条件の範囲を限定していく。

2 つ目としてはループの逆実行を考えなければならない点である。この問題に対しては、制約条件を利用して逆実行を考えることで対応する。図 7 は for(i=0;i<10;i++)というループの逆実行フローである。

ループの逆実行は図 7 の矢印を 1、2、3、4 の順番に進むことになるが制約条件による範囲の絞込行うことで i の値を一意に決めることが可能である。

まず図7の end から始まることになるが、この時 i は任意の値があり得るが、i が矢印 1 を進むと、「i　≧　10　」という制約条件が付くことになる。

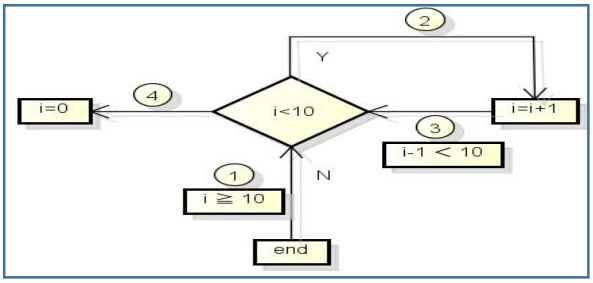

図7. ループの逆実行フロー

次に矢印 2 を進むと i=i+10の命令にたどり着くが、ここでは逆実行なので i=i-1の命令を実行したと考えられる。矢印 2 を進んだ後は最初の end の状態から-1した状態になり、矢印3を進むと i に制約として「i -1＜10」という制約がさらに加わることになる。判明した2つの条件から「10 ≦ i＜11」という条件が得られ、この条件を満たすような i は10なので最初の i は10に限定される。

3 つ目としては条件分岐の逆戻り先が不明な点である。命令を逆方向に辿ることになるので条件分岐の条件式が真だったのか、偽だったのかが不明なためである。この対策としては縦型探索を利用して命令を逆方向から辿ることで対応する。流れとしては図 8 のようになる。

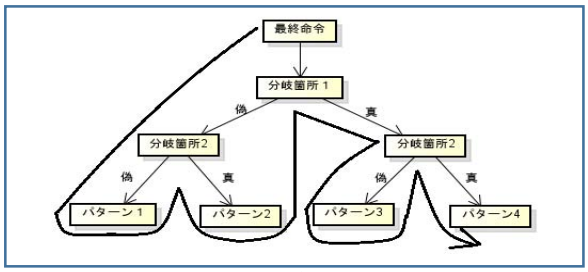

図 8. 分岐処理

最終命令から逆方向に命令を辿っていき、分岐箇所を発見したら、分岐箇所と分岐先の命令番号をスタックに保存する。それを続けていき最初の命令まで辿り着いたらスタックに積んである命令箇所まで戻り、辿っていない分岐先の逆解析を行っていく。

3.研究成果

Java 機械語命令を逆方向解析するプログラムを開発し、想定外とした出力となる入力条件を導くことに成功したが、まだ課題が残っている。具体的には入力値が関係している条件分岐への対応、ループの途中抜けへの対応などである。

4.おわりに

Java プログラムの逆方向解析を提案し、逆方向解析を行うプログラムの開発を行った。今後は残った課題を解決していき、性能の向上を目指す。

5.謝辞

本研究は JSPS 科研費　15K11989　の助成を受けたものです。

6.参考文献

[1]Yukio Hiranaka, Houjin Sakaki, Kenta Ito, Toshihiro Taketa and Shinichi Miura, “Numerical Backward Simulation Model with Case Branching Capability”, SIMULTECH,　2014.

[2]Tim Lindholm, Frank Yelin, 村上雅章訳, ”Java 仮想マシン仕様　第 2 版”, (株)ピアソン・エデュケーション, 2001.