B-004

プログラムの振る舞いの差異を用いたデバッグ支援
Debugging Support Using Differences of Program Behavior

鈴木 貴治† 酒井 三四郎‡
Takaharu SUZUKI Sanshiro SAKAI

1 背景と目的

デバッグはソフトウェア開発において、非常にコストと時間がかかるプロセスである。デバッグは、大きく分けて、誤りのある命令の探索、誤っている命令の修正の２つのプロセスに分けられるが、特に誤りがある命令の探索にコストと時間がかかってしまうと考えられる。よって、バグ探索におけるコストと時間を短縮させることができれば、デバッグにおける時間を短縮させ、コストも減らすことができ、ソフトウェア開発全体におけるコストと時間も大幅に短縮させることができる。

そこで本研究では、正しい結果となる (成功する) ときのプログラムの振る舞いと、誤った結果となる (失敗する) ときのプログラムの振る舞いの差から、どの命令が誤りの可能性が高いかを計算することで、誤りがある命令の探索支援を行う。具体的には、成功するときの変数の値、実行された命令と、失敗するときの変数の値、実行された命令の差から、命令毎に誤りの可能性を計算する。

2 関連研究

ある命令が実行されたかどうかという動的情報を用いて、誤り探索を支援している研究 [1] がある。この研究では、成功するテストケースで実行された数と失敗するテストケースで実行された数の比率から、命令毎に誤りの可能性を数値化し、命令を順位付けすることで誤り探索支援を行っている。

また同様に、命令の実行の有無で、命令の可能性を数値化し、命令の順位付けを行っている研究 [2] がある。この研究が上記の研究と異なる点は、失敗するテストケースで実行された数と、失敗するテストケースの総数と実行された数の比率から、命令の順位付けを行っていることである。この研究では様々な式を挙げて、それぞれの式を比較して、Ochiai という式が最も高いパフォーマンスをあげていることを示している。

3 本研究の位置づけ

前章で紹介した研究はどちらも命令が実行されたかどうかという動的情報を用いて、命令毎に誤りの可能性を計算している。しかし、命令が実行されたかどうかという動的情報だけでは、命令毎の誤り度に差があまりできず、誤りの可能性が同じ命令が多くなってしまう。

そこで、本研究では、引数の値、大域変数の値、関数の返り値も用いることで、他の研究よりも探索範囲を狭め、精度を向上させることを目指す。また、前章で述べた２つの研究と同様に、C 言語を対象言語とし、対象者はプログラミング経験初級者から上級者とする。

†静岡大学大学院情報学研究科
‡静岡大学情報学部

4 システムの概要

4.1 全体の流れ

ユーザに正しい結果となったときのテストケース、誤った結果になったときのテストケースを用意してもらう。システムはこれを受け取り、ユーザが作成したプログラムを、ユーザから与えられたテストケースで実行し、動的情報を取得する。取得するべき動的情報は、引数の値、大域変数の値、関数の返り値、命令の実行の有無である。動的情報は、システムが GDB[1] から取得する。

正しい結果となったときのプログラムの動的情報と、誤った結果となったときのプログラムの動的情報から、命令毎にどれくらい誤りの可能性があるかを計算し、それをユーザに示す。

4.2 命令の誤りの可能性を計算

実行の有無から誤りの可能性を計算するプロセスと変数の値から誤りの可能性を計算するプロセスの 2 つのプロセスに分ける。実行の有無から計算された結果と変数の値から計算された結果の合計を命令の誤り度として、数値が高いほうから、誤りの可能性が高いことを表している。

実行の有無から誤りの可能性を計算する

成功するテストケースの中で、その命令が何個のテストケースで実行されたか、失敗するテストケースの中で、その命令が何個のテストケースで実行されたかを数える。これを式 1 で計算することにより、その命令の誤り度を計算する。failed は失敗するテストケースで実行された数、totalfailed は失敗するテストケースの総数、passed は成功するテストケースで実行された数を指している。この式の結果と、変数の値から求めた誤り度を使用して命令毎に誤り度を求めるが、実行の有無から求めた誤り度に重みをつけるため、式 1 の結果に 9 をかけた結果を使用する。

$$E = \frac{failed \times failed}{totalfailed \times (passed + failed)} \quad (1)$$

変数の値から誤りの可能性を計算する

変数毎に、表 1 のようにまとめる。passed は成功するテストケースでの実行、failed は失敗するテストケースでの実行を表しており、変数 x に値 0 が代入されていた時は、2 つの成功するテストケースで実行されたときであることを示している。

式 1 の結果が最も高くなる組み合わせでの値を変数の誤り度とする。表 1 では、値 1 と値 2 の組み合わせのときに passed が 1、failed が 3 となり、組み合わせの中で最も高くなる 0.75 となるので、変数 x の誤り度は 0.75 となる。

[1]GNU が提供するフリーのデバッガ

表 1: 変数 x の誤り度

値	passed	failed
0	2	0
1	1	2
2	0	1

変数の誤り度を命令に反映させる

次に、変数の誤り度を、命令に反映する。変数 x を参照していると考えられる命令、または、変数 x に値を代入したと考えられる命令に、変数 x の誤り度を加算する。

関数毎に、引数の誤り度、返り値の誤り度、呼び出した関数への引数の誤り度、呼び出した関数の返り値の誤り度、大域変数の誤り度を求める。このとき、返り値の誤り度、呼び出した関数への引数の誤り度へは 2 をかけることで重みをつける。これらの誤り度の平均を求め、その値をその関数に存在する全ての命令の誤り度に加算する。

5 評価

評価は、ベンチマークプログラムセットである Siemens suite[3] を用いて、他の研究との比較を行う。

Siemens suite は、複数のプログラムがあり、それぞれに複数のバージョンが存在する。それぞれのバージョンには、他のバージョンとは異なるバグが埋め込まれており、それに対するテストケースも用意されている。

この中から、セグメンテーション違反が生じないバージョンのみを取り出して、7 つのプログラム、98 のバージョンを使用して、テストケースは 20 個で実験を行う。テストケースの内容は、成功するテストケース 10 個、失敗するテストケース 10 個である。評価には、探索範囲辺りに誤りがある命令が存在する確率を計算して、それらの平均値を用いる。

表 2 は実験結果である。プログラム全体の 1%を調べると、Tarantula では、2.8%の確率で誤りがある命令が存在し、Ochiai では、8.4%の確率で存在し、本システムでは、10.9%の確率で存在することを表している。図 1 は表 2 をグラフ化したものである。

表 2: 評価実験の結果

探索範囲 (%)	Tarantula	Ochiai	本システム
1	2.9	8.5	13.3
10	29.3	48.6	49.8
20	49.4	64.6	66.3
30	63.9	74.6	76.8
40	74.6	80.3	84.8
50	81.2	85.5	90.7
60	86.8	90.2	98.7
70	90.6	93.9	99.0
80	95.1	97.6	100
90	98.4	99.9	100
100	100	100	100

結果から、全体的に本システムが既存技術より上回っていることが分かる。特に探索範囲 51%からは既存の技術より精度が高く、最悪の場合でも、プログラム全体の 80%を調べれば誤りが存在することになる。

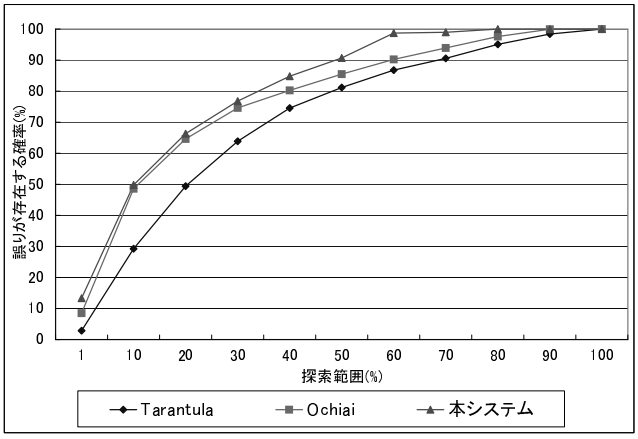

図 1: 評価実験の結果

しかし、50%までは、既存の技術よりも少し精度が上がった程度であり、精度が大幅に向上しているわけではないという問題点が残った。

6 まとめ

評価実験から、本システムが既存技術より上回っていることが確認でき、変数の値、関数の返り値などの動的情報を用いることで、誤り探索支援が行えることを示せた。しかし、既存技術より大幅に精度が向上したわけではなく、精度が高いとは言えない。今後の課題として以下の 3 つが挙げられる。

- 様々な状況での実験
- 精度の向上
- 他の動的情報の活用方法の考案

今回は、多くても 600 行程度のプログラムで実験を行ったが、より大規模なプログラムではどのような結果になるかを実験する必要がある。また、今回扱った動的情報をより活かす方法を考えるとともに、他の動的情報も活用する方法を考えることで、更なる精度の向上を目指す。

参考文献

[1] James A. Jones, Mary Jean Harrold, “Empirical evaluation of the tarantula automatic fault-localization technique”, 20th IEEE/ACM international Conference on Automated software engineering, pages 273-282, 2005.

[2] R.Abreu, P.Zoeteweij, A.J.C. van Gemund, “An Evalution of Similarity Coefficients for Software Fault Localization”, In Proceedings of the 12th International Symposium on Pacific Rim Dependable Computing(PRDC’06), pages 39-46, 2006.

[3] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, “Experiments on the effectiveness of dataflow and controlflow based test adequacy criteria”, 16th International Conference on Software Engineering, pages 191-200, 1994.