# 条件式に欠陥を含むプログラムを対象に SBFL を実施した場合に欠陥箇所を特定する手法

2019SE057 谷恭輔
指導教員：名倉正剛

## 1 はじめに

近年ではソフトウェア開発におけるデバック作業支援として，Spectrum-Based Fault Localization(SBFL) に関する研究が盛んに行われている．本研究では，条件式に欠陥を含むプログラムを対象に SBFL を実施した場合に欠陥箇所を特定する手法を提案する．

## 2 Spectrum-Based Fault Localization (SBFL)

SBFL は，テストケースによる実行経路情報を利用するフォルトローカライゼーション技術である [1]．テストケースごとの成否と，実行経路情報から，各ステートメントに対して疑惑値を算出し，その疑惑値によりプログラム中の欠陥箇所を推定する．テストケースによる実行経路情報を利用し，各ステートメントに対して疑惑値の算出をした結果，疑惑値が高いステートメントほど障害原因箇所である可能性が高いものとして扱う．

## 3 課題

### 3.1 本研究で対象にする課題

本研究では，成否が異なるにもかかわらず実行経路が同じであるテストケースが存在する場合の障害原因箇所の特定を課題とする．条件式に欠陥がある場合に，疑惑値の算出結果によって障害原因箇所を特定することが困難になる場合があり，その例として SBFL を利用した各ステートメントに対する疑惑値算出の表 1 を示す．この例は，1 桁の自然数かどうかを判定する関数で，自然数なら 0 を，0 以下なら −1 を，2 桁以上なら 1 を返す．しかし，$s2$ に欠陥箇所が存在しており，本来なら条件式が x <= 0 であるところを x < 0 に間違えている．

**表 1:** 課題の例

| プログラムコード | test1 (x=-1) | test2 (x=0) | test3 (x=9) | test4 (x=10) | 疑惑値 |
|---|---|---|---|---|---|
| $s1$: int SingleDigit(int x) { | | | | | |
| $s2$: if (x < 0) //correct:x <= 0 | ✓ | ✓ | ✓ | ✓ | 0.50 |
| $s3$: return -1; | ✓ | | | | 0.00 |
| $s4$: if (x >= 10) | | ✓ | ✓ | ✓ | 0.58 |
| $s5$: return 1; | | | | ✓ | 0.00 |
| $s6$: return 0; | | ✓ | ✓ | | 0.71 |
| $s7$: } | | | | | |
| 期待値 | -1 | -1 | 0 | 1 | |
| 実際の値 | -1 | 0 | 0 | 1 | |
| 成否 | P | F | P | P | |

表 1 の例では，test1 から test4 はそれぞれ x= −1，0，9，10 の異なるテストケースとそれに対する実行経路情報を表しており，実行された行に対して ✓ を示している．また，実行経路情報から，期待値と実際の値で成否を判定し，成功なら P，失敗なら F を示した．疑惑値の列には Ochiai の計算式により算出した $s2$ から $s6$ までのそれぞれのステートメントに対する疑惑値算出結果を示している．そもそも，この例では test2，test3 は成否が異なるにもかかわらず実行経路が同じであり SBFL では本質的に特定できない．よって，はじめに述べたように，成否が異なるにもかかわらず実行経路が同じであるテストケースが存在する場合の障害原因箇所の特定を課題とする．

## 4 提案手法

### 4.1 概要

本研究では，3.1 節で述べた課題を解決するため，条件式に欠陥を含むプログラムを対象に SBFL を実施した場合に欠陥箇所を特定する手法を提案する．
3.1 節で述べたように，同じ実行経路で実行される成功テストケースと失敗テストケースが存在すると，欠陥の存在するステートメントを通る失敗テストケースと成功テストケースの数に差がなくなることがあり，その場合には障害原因を特定できなかった．ここで仮に条件式の評価の真偽が逆になるようにテスト対象のプログラムを書き替えると，成功する実行については分岐によって実行される処理が異なるので，失敗するようになる．逆に失敗する実行について同様に条件式を書き替えた場合は，仮に書き替えた条件式が欠陥箇所であるならば，成功するようになる．したがって失敗テストケースの実行経路に含まれる条件式を一つずつ順に真偽が逆になるように書き替えた場合に成功するように変化するならば，書き替えた場所が欠陥箇所である．本研究ではこのような考えに基づき，失敗テストケースの実行経路上の条件式に対して真偽を逆にするようなミュータントを生成して実行したテスト結果を利用することで，欠陥箇所を特定する手法を提案する．

### 4.2 手順

障害原因箇所の特定の手順を図 1 に示す．

1. プログラムとテストケース群を入力とし，SBFL ツールによる処理を実施したときに，SBFL 結果と実行経路情報が出力される．
2. 1 の出力で得られた実行経路情報を入力とし，実行経路解析処理を実施したときに，失敗するテストケースに対する実行経路情報が出力される．実行経路解析処理では，実行経路情報に記録されている 1 つ目のテストケースから順に実行経路情報を比較し，同じパスを通るが成否が異なる実行経路情報のうち，失敗するテストケースに対する実行経路情報を抽出する．

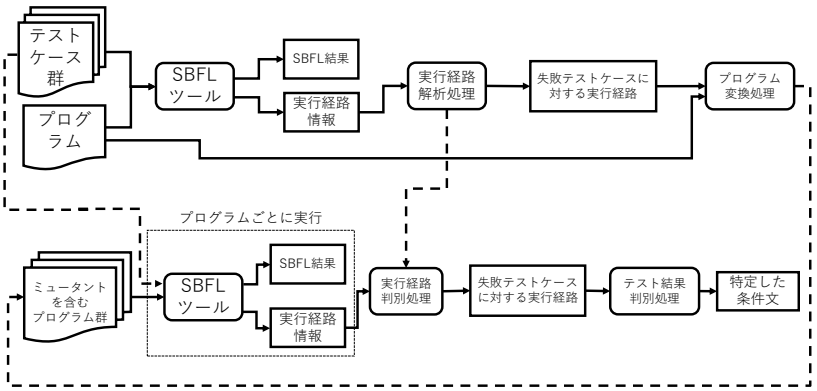


**図 1:** 障害原因特定の手順

3. プログラムを入力とし，プログラム変換処理を実施したときに，ミュータントを含むプログラム群が出力される．プログラム変換処理では，プログラム中の条件式に対してミュータントを付加する．
4. 1 と同様に，ミュータントを含むプログラム群とテストケース群を入力とし，SBFL ツールによる処理を実施したときに，SBFL 結果と実行経路情報が出力される．
5. 4 で得た実行経路情報と 2 で得た失敗テストケースに対する実行経路情報を入力とし，実行経路判別処理を実施したときに，失敗テストケースに対する実行経路情報が出力される．入力とする実行経路情報に含まれる，テストケースがそれぞれ一致することで失敗テストケースの実行経路情報が出力される．
6. テスト結果判別処理では，5 で得た失敗するテストケースに対する実行経路情報を入力とし，このとき実行経路情報の成否に着目することで，成功するように変化している実行経路情報が存在すれば，その実行経路情報に対応するミュータントを含むプログラムが出力される．その結果，プログラム中に存在するミュータントを含む if ステートメントが障害原因箇所として特定される．

### 4.3 障害原因箇所特定の実施例

表 2 にミュータントに対する実行結果の例を示す．この例では，表 1 で示したテストケースのうち，4.2 節の手順より，失敗するテストケースである test2 に対して障害原因箇所の特定をする．表 2 はテストケースの通る経路上である $s2$ に条件式の真偽を反転させるミュータントを生成している．その結果，期待値と実際の値が一致することを確認でき，テストが成功することでテストの成否が変わることがわかる．また $s4$ にも同様にミュータントを生成し実行すると，この例ではテストケースの成否は変わらない．よって，$s2$ のステートメントが障害原因箇所であると特定できる．

**表 2:** ミュータントに対する実行結果の例

| プログラムコード | test2 (x=0) |
|---|---|
| $s1$: int SingleDigit(int x) { | |
| $s2$: if (!(x != 0)) | ✓ |
| $s3$: return -1; | ✓ |
| $s4$: if (x >= 10) | |
| $s5$: return 1; | |
| $s6$: return 0; | |
| $s7$: } | |
| 期待値 | -1 |
| 実際の値 | -1 |
| 成否 | P |

## 5 おわりに

本研究では，条件式に欠陥を含むプログラムを対象に SBFL を実施したときに欠陥箇所を特定することを目的に，1 つの対象とするプログラムとテストケースを入力としたときに欠陥を含む条件式を特定することができた．

## 参考文献

[1] W.E. Wong et al., A Survey on Software Fault Localization, IEEE Trans. on Software Engineering, Vol. 42, No. 8, pp. 707-740, 2016.

[2] B.J. Choi et al., The Mothra tool set (software testing), Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences, Vol. 2, pp. 275-284, 1989.

[3] M. Papadakis et al., Metallaxis-FL: mutation-based fault localization, Software Testing Verification and Reliability, Vol. 25, pp. 605–62, 2015.

[4] J. Xuan et al., Nopol: Automatic Repair of Conditional Statement Bugs in Java Programs, IEEE Trans. on Software Engineering, Vol. 43, No. 1, pp. 34-55, 2017.