JavaScriptを対象とした スペクトラムに基づく欠陥限局ツールの試作

小田 郁弥[1,a)]　柗本 真佑[1,b)]　楠本 真二[1,c)]

概要：デバッグ作業を支援する技術の一つに，欠陥限局と呼ばれる技術が存在する．欠陥限局手法のうち，近年盛んに研究されているものとして，Spectrum-Based Fault Localization（以下，SBFL）と呼ばれる手法が存在する．SBFL は，テストケースの実行経路からプログラムの欠陥箇所を推定する技術である．SBFL では，実行経路の取得方法が言語ごとに固有であるなどの理由により，各言語ごとに専用の SBFL ツールが必要とされる．また，バックエンドでの活用が盛んにもかかわらず，SBFL ツールが実装されていない言語として JavaScript が存在する．本研究では，JavaScript 用 SBFL ツールを Node.js のモジュールとして実現する．実現にあたって，そのプロトタイプとして InagoFL を実装した．12 個の人為的に挿入した欠陥と，10 プロジェクトに含まれる 453 個の欠陥への適用実験の結果，InagoFL の欠陥検出能力と適用可能性を確認することができた．

Prototyping a Spectrum-Based Fault Localization Tool for JavaScript

1. はじめに

デバッグ作業を支援する技術の一つに，欠陥限局と呼ばれる技術がある．欠陥限局とは，プログラム中に発生した欠陥の原因箇所を推定する技術である．これまでにも様々な欠陥限局手法が提案されている [1] [2] [3]．その中でも Spectrum-Based Fault Localization（以下，SBFL）という手法が存在する [3]．SBFL は近年盛んに研究されている欠陥限局の手法の一つである．SBFL は，テストケースの実行経路からプログラムの欠陥箇所を推定する技術である．失敗したテストケースで実行したプログラム文は欠陥の可能性が高く，成功したテストケースで実行した文が欠陥である可能性は低いというアイデアに基づき，欠陥箇所の推定を行う．SBFL の応用は多岐にわたり，人間が目視で欠陥を修正する際の支援や，自動プログラム修正の分野で応用されている [4]．

SBFL は上記の活用を可能にする一方で，SBFL を行うツールは複数のプログラミング言語に対応できないという問題を抱えている．SBFL は欠陥位置の推定にテストケースの実行経路を必要としており，実行経路の測定方法は各プログラミング言語ごとに異なっているためである．この理由によって，SBFL を行うツールが未実装であるプログラミング言語が存在する [3]．

SBFL を行うツールが未実装である言語として JavaScript が存在する．JavaScript は近年の Web プログラミングの中心となる言語であり，フロントエンドでの活用だけでなく，バックエンドでの利用も広がっている．また，バックエンドでの利用に際して，単体テストを行う文化も存在している．しかし，SBFL を行うツールが存在しないため，先述したような活用が不可能である．

本研究の目的は JavaScript におけるデバッグ作業の支援である．そのために，JavaScript の SBFL ツールを作成する．実現にあたり，その試作として InagoFL を実装した．これは JavaScript 用テストフレームワークの Mocha で記述されたテストを実行して SBFL を行う Node.js 用モジュールである．

InagoFL の欠陥検出能力を確認するため，ミューテーションツールを用いて作成された 12 個の欠陥を含むプログラムに対して SBFL を行った．その結果，全ての欠陥に対して SBFL を行えることを確認した．InagoFL の適用可能性を確認するため，JavaScript の実プロジェクト 10 プロ

[1] 大阪大学大学院情報科学研究科
Osaka University
[a)] fumy-oda@ist.osaka-u.ac.jp
[b)] shinsuke@ist.osaka-u.ac.jp
[c)] kusumoto@ist.osaka-u.ac.jp

ジェクトに含まれる 453 個の欠陥に対して適用実験を行った．その結果，8 プロジェクトで動作を確認し，403 個の欠陥で正常に動作することを確認した．加えて，98 個以上の 0 より大きい疑惑値が付与された欠陥に対し，欠陥限局に用いた計算式による精度を比較することで，JavaScript の実プロジェクトに含まれる欠陥に対する SBFL の精度について確認した．その結果，Ample と呼ばれる計算式を用いて SBFL を行うと，精度の高い欠陥限局が行えることを確認した．

2. 準備

2.1 SBFL

ソースコードに含まれる欠陥の原因箇所の特定を目的とした，欠陥限局に関する研究が多数行われている [1] [2] [3]．欠陥限局とは，過去の欠陥情報やテストの実行結果等を用いて，欠陥の原因箇所を推定する技術である．欠陥限局手法にはプログラムのスライスを用いる手法 [1] や実行中のプログラムの変数の値を用いる手法 [2] など，様々な手法が提案されている．

欠陥限局のうち，テストの実行情報を用いるものとして SBFL が存在する [3]．SBFL におけるスペクトラムとは，どのプログラム文が実行されたかという実行経路情報である．SBFL の基本アイデアは以下の通りである．失敗テストケースで実行されたプログラム文は欠陥の原因箇所である可能性が高く，成功テストケースで実行されたプログラム文は欠陥の原因箇所である可能性が低い．

SBFL では，あるプログラム文が欠陥の原因箇所である可能性を疑惑値という値で表す．疑惑値はプログラムの各行に付与され，大きな疑惑値が付与された行ほど欠陥を含む可能性が高いと推測される．疑惑値の具体的な算出方法を図 1 を用いて説明する．この図は 3 行目に欠陥を含む FizzBuzz プログラムであり，3 つのテストケースを持っている．各列の○は各テストで実行された行を示している．3 行目の欠陥により，テスト 1 とテスト 2 は通過しているが，テスト 3 は通過していない状況である．疑惑値の算出にあたっては，以下の 4 つの値を考える．

- $fail(s)$:行 s を実行する失敗テストケースの数
- $pass(s)$:行 s を実行する成功テストケースの数
- $totalFail$:失敗テストケースの総数
- $totalPass$:成功テストケースの総数

上記 4 つの指標に基づいて，行 s の疑惑値を算出する．疑惑値の算出の為の式（以下，SBFL 式）には様々な式が提案されている．ここでは式 1 に示す Ochiai と呼ばれる SBFL 式 [5] を用いて説明する．

$$susp(s) = \frac{fail(s)}{\sqrt{totalFail * (fail(s) + pass(s))}} \quad (1)$$

1 行目の疑惑値である $susp(1)$ について考える．1 行目

	テスト1 (n=3) ✓	テスト2 (n=5) ✓	テスト3 (n=15) ✗	疑惑値
1:function FizzBuzz(int n) {	●	●	●	0.6
2: if (n % 15 == 0)	●	●	●	0.6
3: s = "";			●	1.0
4: else if (n % 3 == 0)	●	●		0.0
5: s = "fizz";	●			0.0
6: else if (n % 5 == 0)		●		0.0
7: s = "buzz";		●		0.0
8: return s;	●	●	●	0.6
9:}				0.0

図 1 欠陥を含む FizzBuzz プログラム

を通過した失敗テストの数が 1 つ，成功テストの数が 2 つである．よって，$fail(1) = 1$，$pass(1) = 2$ であるので，$susp(1) = 0.6$ である．同様に他の行を計算することで，欠陥が含まれている可能性を比較することができる．3 行目は失敗テストのみが通過しているので，$susp(3) = 1$ であり，4 行目は成功テストのみが通過しているので $susp(4) = 0$ である．疑惑値の大小で判断すると，4 行目より 1 行目に欠陥が含まれる可能性が高く，1 行目より 3 行目に欠陥が含まれる可能性が高いと解釈できる．このように，テストケースの成否と，通過した情報をもとに疑惑値を算出することで，欠陥の箇所をある程度絞り込むことができる．

表 1 に示すように，SBFL 式には Ochiai 以外にも様々な式が提案されている．これらの SBFL 式はそれぞれ異なる値を出力する．そのため，各行の疑惑値の大小は使用する SBFL 式によって変化し，欠陥限局の精度を左右する．複数存在する SBFL 式の欠陥検出精度の優劣の評価を目的とし，Abreu らが比較実験を行っている．この実験では C 言語で記述されたプログラムの欠陥に対して，Ochiai で算出された疑惑値が最も効果的に欠陥限局を行えることを報告している [6]．また，Wong らの研究では，C 言語と Java における欠陥限局では，DStar が他の 31 個の SBFL 式と比較して優れているという結果を示している [7]．

2.2 JavaScript

JavaScript はフロントエンドの Web プログラミングで広範に使用される言語である．フロントエンドでの使用例としては Google マップをはじめとする Ajax Web アプリケーションが挙げられる．

JavaScript は当初，フロントエンドで使用されていたが，

表 1 SBFL 式の例

式名	定義
Ample	$\lvert\frac{fail(s)}{totalFail} - \frac{pass(s)}{totalPass}\rvert$
DStar(*=N)	$\frac{fail(s)^N}{(totalFail-fail(s))+pass(s)}$
Jaccard	$\frac{fail(s)}{totalFail+pass(s)}$
Ochiai	$\frac{fail(s)}{\sqrt{totalFail*(fail(s)+pass(s))}}$
Zoltar	$\frac{fail(s)}{totalFail+pass(s)+\frac{10000*(totalFail-fail(s))*pass(s)}{fail(s)}}$

バックエンドで JavaScript を使用するための実行環境である Node.js が登場し，以降バックエンドでの活用も広がっている．JavaScript のバックエンドでの活用は多岐にわたり，MySQL を用いたデータベース構築や，PDF の作成，更にはソーシャルゲームをはじめとしたアプリのサーバ等でも活用されている．

Node.js では Mocha をはじめとした単体テストのフレームワークも開発されている．標準規格として ECMAScript が定められており，様々な実装が存在する．JavaScript の実装の中で広く使われているエンジンとして V8 が存在する．V8 は Google Chrome を代表とする Chromium ベースのブラウザや，先述した Node.js で使用されている．

2.3 既存の SBFL ツール

SBFL については複数の先行研究が存在しており，研究成果として Zoltar [13] や CharmFL [9] をはじめとした，オープンソースのツールも多数提供されている．既存の SBFL ツールの一覧を表 2 に示す．Wong らの調査 [3] によると，SBFL を含む欠陥限局ツールは C 言語や Java を対象としたものが多く，SBFL ツールにも同様の傾向が見られる．また，同調査では JavaScript の SBFL ツールの存在に言及しておらず，我々の知る限り存在していない．

SBFL の活用は多岐に渡る．具体的には人間が目視で欠陥を限局する際の優先度の指標としての活用や，自動プログラム修正技術と組み合わせ，限局した欠陥の自動修正を行うなどが挙げられる．しかし，JavaScript においては SBFL ツールが現状提供されていないため，上記のような活用は不可能である．

3. 提案ツール:InagoFL

3.1 概要

本論文では JavaScript におけるデバッグ作業の支援を目的とし，JavaScript を対象とした SBFL ツール InagoFL を提案する．InagoFL が使用可能な JavaScript プロジェクトは，Node.js で動作する，Mocha でテストケースが記述されたプロジェクトである．InagoFL は現在，表 1 に示した 5 つの SBFL 式で欠陥限局が可能である．また InagoFL は OSS として公開されている[*1]．

表 2 既存の SBFL ツール

ツール名	対応言語
Apollo [8]	PHP
CharmFL [9]	Python
GZoltar [10]	Java
Jaguar [11]	Java
TARANTULA [12]	C
Zoltar [13]	C

[*1] https://github.com/OdaFumiya/InagoFL

3.2 処理の流れ

ユーザーが InagoFL を実行すると，Mocha で記述されたテストケースが個別に実行され，各テストケースの成否と経路情報が出力される．テストケースの実行は Node.js に用意されている子プロセス生成関数である execSync からシェルコマンドを実行することによって行われる．

経路情報の取得は，V8 エンジン用カバレッジ取得ツール c8 によって行われる．経路情報と成否は，$fail(s)$ や $pass(s)$ などの，欠陥限局に必要な値に変換する．これらの値を用いて指定された SBFL 式で欠陥限局を行い，各行の疑惑値を出力する．また，テストケースの経路情報と成否を再利用する場合は，テストケースの実行を省略し，疑惑値を算出することもできる．これは，テストケースの実行結果が変化しない場合の実行時間の短縮や，Flaky Test と呼ばれる実行結果が不安定なテストケースの経路情報や成否を変化させずに，同一の結果を再利用して欠陥限局を行う際に利用できる．

InagoFL はテストケースの実行の際，InagoFL 内での実行と同様の処理を行うシェルスクリプトを出力する．出力されたスクリプトを参照することで，特定のテストケースのみの再実行や，新しく追加したテストケースのみの実行，特定のテストケースのみを実行しないといったユーザーの要求にも対応できる．

また，テストケースの実行が行われなかった場合や，実行時間がタイムアウトしたテストケースなど，Mocha においてテストケースの成否が確定しない場合，該当テストケースの情報は疑惑値の算出に用いられない．これは $pass(s)$ と $totalPass$ の算出を全テストケースの個数から失敗テストケースの個数を減算することで行っているため，成否が確定していないテストケースが成功テストケースとして扱われ，正確な疑惑値の算出が行われなくなるためである．

3.3 利用の流れ

利用の際は“`inagoFL`”とコマンドラインに入力とテストケースが実行される．InagoFL のインストールには Node.js 用パッケージ管理ツール npm を用いる．

表 3 に InagoFL のオプションを示す．InagoFL で欠陥限局を行う際は，オプションから SBFL 式を指定する必要がある．加えて，出力ファイル名の指定や，疑惑値の計算で用いたスペクトラムの出力などもオプションから操作できる．また，InagoFL の利用に際してテストケースやソースコードに前処理を行う必要はない．

InagoFL を用いた欠陥限局の方法を具体例を挙げて説明する．Ochiai の SBFL 式で疑惑値を計算する場合，“`inagoFL --Ochiai`”とコマンドラインに入力することで，Ochiai の SBFL 式で計算された疑惑値が JSON ファイルが出力される．また，その際にディレクトリが作成され，各テスト

ケースごとの実行経路情報と成否が保存される．これらの情報は InagoFL でテストを実行すると更新される．各テストケースごとの実行経路情報と成否を再利用して他の疑惑値を算出する場合は“-s”オプションを用いる．“inagoFL -s --Ample”と入力することで，直前に用いた結果を再利用して Ample の式で計算された疑惑値を出力することができる．

4. 欠陥検出能力の実験

4.1 概要

InagoFL の欠陥検出能力を確認することを目的に実験を行った．欠陥を人為的に挿入したプログラムに対し，InagoFL が SBFL によって欠陥限局が可能であるかを確かめる．

題材は図 2 に示した 10 行程度の整数の加減算を行うプログラムを対象とする．JavaScript で記述された対象プログラムに対して，網羅率 100%の Mocha で記述されたテストケースを用意し，プログラムに人為的に欠陥を挿入し，InagoFL で欠陥限局を行う．

欠陥の挿入には JavaScript 用ミューテーションテストフレームワーク StrykerJS[*2]を用いる．上記題材の場合，表 4 に示される 12 の箇所に欠陥が挿入された．この表は StrykerJS が欠陥を挿入した際に，図 2 のプログラムのどの部分が改変されたかを示している．欠陥 ID#1 は，図 2 のプログラムの 3 行目の“n>0”が“true”に書き換えられたことを示す．また，欠陥 ID#5 のように，欠陥の挿入の為に文が削除される場合も存在する．

欠陥検出能力の計測指標として topN%を用いる．これは欠陥箇所に付与された疑惑値が全体の上位 N%以内に入っているかを評価する指標である．欠陥限局の精度が高い場合，欠陥箇所に付与された疑惑値は他の箇所と比較して高くなり，順位は上位になると言える．即ち，小さい N の値に対し，上位 N%以内の順位が付けられた欠陥の数が多いほど，欠陥限局の精度が高いと言える．

また，疑惑値として 0 または null が算出される場合がある．これらの欠陥は検出されなかった欠陥として扱う．

表 3 InagoFL のオプション

オプション	効果
-a, --all	全ての SBFL 式で疑惑値を計算する
-h, --help	ヘルプを表示する
-l, --list	テストケースのリストを出力する
-n, --name “name”	結果を“name”として出力する
-r, --result	テストケースの実行結果を出力する
-s, --skip	テストの実行をスキップする
-t, --test “dir”	“dir”以下のディレクトリのテストを実行する
-v, --version	バージョンを表示する
--Ample	Ample で疑惑値を計算する
--DStar2	DStar(*=2) で疑惑値を計算する
--DStar3	DStar(*=3) で疑惑値を計算する
--DStar10	DStar(*=10) で疑惑値を計算する
--Jaccard	Jaccard で疑惑値を計算する
--Ochiai	Ochiai で疑惑値を計算する
--Zoltar	Zoltar で疑惑値を計算する

[*2] https://stryker-mutator.io/

```
 1:function ctz(n){
 2:   var ret = n;
 3:   if(n>0){
 4:     ret--;
 5:   }
 6:   if(n<0){
 7:     ret++;
 8:   }
 9:   return ret;
10:}
```

図 2 欠陥検出能力の実験に用いたプログラム

4.2 結果

各 SBFL 式で順位が上位 N%以内の欠陥数を図 3 に，疑惑値として 0 または null が付与された欠陥を表 5 に示す．表 5 の欠陥 ID は表 4 の欠陥 ID と同一のものであり，SBFL 式は該当する欠陥に 0 または null の疑惑値を付与した SBFL 式である．この実験において，InagoFL が実行できなかった欠陥や，いずれの SBFL 式でも 0 より大きい疑惑値が付与されなかった欠陥は存在しなかった．

実験の結果，Jaccard や Ochiai，Zoltar は，すべての欠陥に対して上位 30%以内の疑惑値を付与している．また，DStar 以外のすべての FL 式はすべての欠陥に対して上位 50%以内の疑惑値を付与している．0 または null の疑惑値を付与した欠陥が存在する SBFL 式は Ample と DStar である．

4.3 考察

表 5 に示した，検出されなかった欠陥について，原因

表 4 欠陥が挿入された箇所

欠陥 ID	改変行	改変前	改変後
#1	3	n>0	true
#2	3	n>0	false
#3	3	n>0	n>=0
#4	3	n>0	n<=0
#5	4	ret--	(削除)
#6	4	ret--	ret++
#7	6	n<0	true
#8	6	n<0	false
#9	6	n<0	n>=0
#10	6	n<0	n<=0
#11	7	ret++	(削除)
#12	7	ret++	ret--

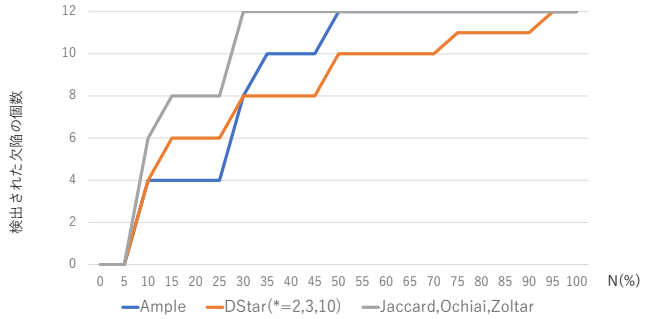

図 3 各 SBFL 式で上位 N%以内に検出された欠陥数

の考察を述べる．今回の実験で検出されなかった欠陥は，Ample の SBFL 式で 10 個，DStar の SBFL 式で 4 個であった．

Ample の SBFL 式で検出されなかった欠陥は#6 と#12 を除くすべての欠陥である．これらの欠陥について，疑惑値を目視で確認した結果，疑惑値として null が付与された欠陥が#9，#5 の欠陥であり，他の欠陥には疑惑値 0 が付与されていた．#9，#4 の欠陥について，疑惑値と経路情報を目視で調査したところ，これらの欠陥は全てのテストケースが失敗しており，Ample の SBFL 式で疑惑値を計算すると，0 除算が発生する．0 除算が行われた結果，Infinity が算出され，JSON ファイルに出力した際に疑惑値として null が付与された．疑惑値 0 が付与された欠陥についても同様に調査を行った．Ample の SBFL 式では，$\frac{fail(s)}{totalFail}$ と $\frac{pass(s)}{totalPass}$ の値が等しくなると疑惑値 0 が算出される．全てのテストケースで通過した行は，両方の値が 1 となるため，疑惑値 0 が算出される．if 文に欠陥箇所が挿入された場合，条件判定の際に通過扱いになる．このために全てのテストケースで該当箇所を通過したと扱われ，疑惑値 0 が算出されたと考えられる．if 文以外の個所に欠陥が挿入された#5，#11 の欠陥について，ソースコードを目視で確認したところ，プログラム文の削除に伴い，該当箇所の改行が削除されていた．これにより，改変箇所が if 文と同一の行となり，同様の理由で疑惑値 0 が算出されたと考えられる．

表 5 0 または null の疑惑値を付与した SBFL 式

欠陥 ID	SBFL 式
#1	Ample
#2	Ample
#3	Ample
#4	Ample,DStar
#5	Ample
#6	DStar
#7	Ample
#8	Ample
#9	Ample,DStar
#10	Ample
#11	Ample
#12	DStar

DStar の SBFL 式で検出でされなかった欠陥は#4，#6，#9，#12 の欠陥である．これらの欠陥について，疑惑値を目視で確認した結果，全ての欠陥に疑惑値 null が付与されていた．Ample の調査と同様に，疑惑値と経路情報を目視で調査したところ，これらの欠陥は全ての失敗テストケースが通り，かつ成功テストケースでは通らない行が存在しており，$(totalFail - fail(s)) + pass(s) = 0$ となるために，DStar の SBFL 式で疑惑値を計算すると 0 除算が発生する．これにより疑惑値として null が付与されたと考えられる．

以上より，検出されなかった欠陥は，SBFL 式または JSON の仕様に起因するものであった．0 除算に起因する null の疑惑値の付与は実装上の問題であり，今後改善するべき課題である．また，他の SBFL 式では $totalFail$ が 0 でなければ 0 除算が起こらず，今回の実験でも 0 や null の疑惑値が付与された行が存在しなかった．このことから，Jaccard，Ochiai，Zoltar の 3 つの SBFL 式において，InagoFL はこれらの式に期待される欠陥検出能力を持つと考えられる．また，Ample と DStar に対しても，0 除算が発生する行以外に対してであれば正しく疑惑値が付与できると考えられる．

5. 実プロジェクトへの適用実験

5.1 実験設定

InagoFL の実プロジェクトに対する適用可能性と，SBFL の JavaScript の実際の欠陥に対する欠陥限局能力を確認することを目的に実験を行った．実験の対象には BugsJS [14] を選択した．BugsJS は Gyimesi らの提供する JavaScript の欠陥データセットである．表 6 にプロジェクト名と各プロジェクトに含まれる欠陥数を示す．BugsJS に含まれる 10 プロジェクトには，合計 453 個の欠陥が含まれる．各欠陥には，欠陥修正前のコミットと欠陥修正後のコミット，欠陥を検出するテストケースを追加した欠陥修正前のコミットが存在する．また，10 プロジェクトは，GitHub におけるスター数 100 以上，コミット数 200 以上，最終コミットが 2017 年以降の Node.js で動作する 10 プロジェクトである．これらの基準から，BugsJS に含まれるプロジェクトは多くの人に利用されるプロジェクトと言える．

適用可能性の評価は，欠陥箇所が発見された 453 個の欠陥に対して InagoFL を適用し，InagoFL が正常終了した欠陥と異常終了した欠陥の割合を計測することで行う．Node.js モジュールはモジュール間の依存関係のエラーなどが原因で，コマンドが異常終了する場合がある．異常終了した場合は疑惑値の算出が行われず，該当しる欠陥に対して InagoFL が適用できなかったと考えられる．また，異常終了した欠陥に対してはその原因を調査する．

欠陥限局能力の評価は，InagoFL が適用可能であった欠

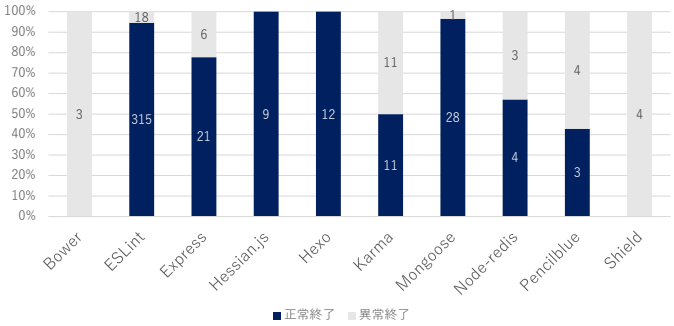

図 4 各プロジェクトで InagoFL が正常終了したものの割合

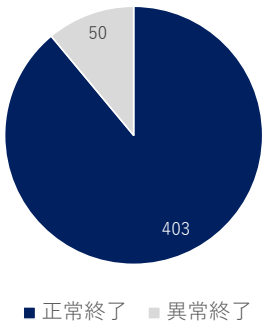

図 5 各欠陥で InagoFL が正常終了したものの割合

陥のうち，欠陥箇所の疑惑値が 0 より大きかった欠陥に対して，topN%を用いて欠陥限局の精度を評価する．これは欠陥を含む行の疑惑値の順位が，疑惑値がついた行全体の上位何%以内に含まれるかという観点で評価を行う．SBFL で精度の高い欠陥限局能力が行えているのであれば，これは欠陥を含む行の疑惑値の順位は小さくなると考えられるためである．欠陥箇所は BugsJS に含まれる各欠陥に対し，欠陥修正前後の差分で最初に変更が加えられた箇所とした．また，25 個の欠陥に対しては差分が検出されなかったため，実験から除外した．

5.2 結果

図 4 に InagoFL を各プロジェクトの欠陥に適用した結果を示す．10 プロジェクト中，8 つのプロジェクトで InagoFL が正常に動作し，疑惑値を算出できることを確認し，Hessian.js と Hexo の 2 プロジェクトに含まれる全ての欠陥で InagoFL の動作を確認した．また，Bower と Shield の 2 プロジェクトでは全ての欠陥に対して異常終了した．

図 5 に全ての欠陥のうち，InagoFL が正常終了した欠陥の割合を示す．453 の欠陥のうち，403 の欠陥で InagoFL が正常終了し，疑惑値を算出することができた．

表 7 に各 SBFL 式で算出した疑惑値が 0 より大きかった欠陥の個数を示す．この数値は Ample の SBFL 式とそれ以外の SBFL 式で異なる結果となった．

図 6 に上位 N%以内に検出された欠陥の割合を示す．N が 5 以上になると，Ample の式が他の SBFL 式よりも高い精度で欠陥を検出している．

表 6 BugsJS に含まれるプロジェクトと欠陥数

プロジェクト名	欠陥数
Bower	3
ESlint	333
Express	27
Hessian.js	9
Hexo	12
Karma	22
Mongoose	29
Node-redis	7
Pencilblue	7
Shield	4

5.3 考察

InagoFL の可搬性

InagoFL が実行されなかった 50 個の欠陥について，原因を目視により確認した．いずれのプロジェクトでも使用モジュールのインストールの際に，パッケージの競合エラーが発生していることが確認された．InagoFL で使用しているモジュールが，各プロジェクトの使用しているモジュールと競合し，必要なモジュールがインストールがなされなかったために InagoFL が停止したものと考えられる．それ以外の場合では InagoFL が停止した欠陥は存在しなかった．

また，Mocha で記述されたテストを Make 等の外部コマンドを用いてテストを実行するプロジェクトに InagoFL を実行したところ，正常に動作するものがあることを確認した．停止したものは先述したパッケージの競合による停止であった．これにより，InagoFL は Mocha で記述されたファイルが存在するならば，実行の方法に依らず欠陥限局が行われると考えられる．

JavaScript の実際の欠陥に対する SBFL の精度

SBFL の欠陥限局精度については Wong らの研究が先行研究として存在する [7]．この研究では，DStar の欠陥検出能力が示されており，DStar の SBFL 式を用いることで，最悪のケースでも疑惑値の順位の上位 30%以内に 60%以上

表 7 疑惑値が 0 より大きかった欠陥数

プロジェクト名	Ample	その他
Bower	0	0
ESlint	71	60
Express	18	17
Hessian.js	0	0
Hexo	8	11
Karma	0	0
Mongoose	6	10
Node-redis	2	0
Pencilblue	0	0
Shield	0	0
合計	105	98

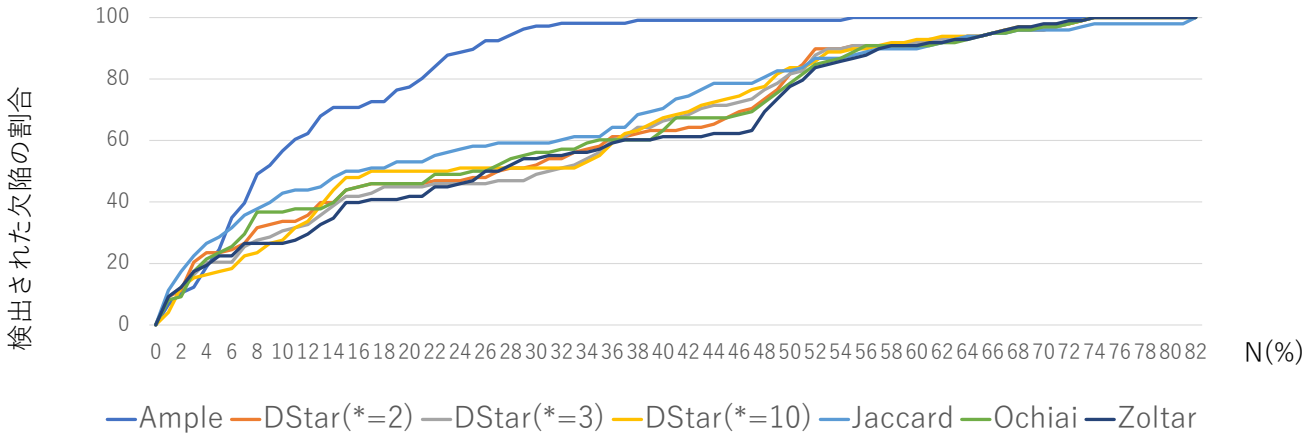

図 6 各 SBFL 式で上位 N%以内に検出された欠陥の割合

の欠陥が検出されるという実験結果が示されている．しかし，今回の実験では，上位 30%以内に DStar が検出された欠陥は 50%前後に留まる結果となった．これは JavaScript の実際の欠陥に対し，DStar の式による欠陥検出能力が他の言語の欠陥に対する欠陥検出能力よりも低い可能性を示唆している．

一方，最も精度が高い欠陥限局が行われた Ample の SBFL 式の検出能力は上位 30%以内に 97%もの欠陥を検出することに成功している．これは同研究の C 言語の欠陥に対する DStar の欠陥検出能力に匹敵する精度であり，Ample の SBFL 式が JavaScript の実際の欠陥の検出に有効である可能性を示している．また，同研究においては Ample の SBFL 式に対する言及は存在しなかった．

SBFL の精度についてのユーザーの意識調査として Kochhar らの調査が存在する [15]．これによると，386 人の利用者に対するアンケートの結果，欠陥箇所の疑惑値が上位 5 位以内に存在した場合，SBFL による欠陥限局が成功したと考えると回答した利用者が 80%を占めると回答している．今回の実験で上位 5 位以内の疑惑値を付与された欠陥の数を目視で確認したところ，Ochiai と DStar(*=10) が 2 個，その他の SBFL 式では 1 個であった．このことから，InagoFL による JavaScript プログラムへの欠陥限局の精度は，利用者にとって十分でないと考えられる．

また，JavaScript の実際の欠陥に対する SBFL の精度の調査結果として Vancsics らの調査が存在する [16]．この研究は BugsJS を題材とし，各種 SBFL 式で欠陥限局を行い，各欠陥箇所の疑惑値が上位の何位以内に含まれるか調査したものである．こちらで調査された SBFL 式は Ochiai, DStar，Tarantula の 3 式である．いずれの式でも上位 10 位以内に 90%程度の欠陥が検出されており，本論文の実験結果と異なる結果を得ている．異なる結果を得た理由として，以下の二つの理由が考えられる．一つは欠陥箇所の設定である．本研究では欠陥箇所を欠陥修正前後の差分における最初の変更箇所とした．これらの変更箇所には，変数の追加などの欠陥の修正に寄与しない変更も含まれる．そのため，本来失敗テストが通過しない箇所が欠陥箇所と判断された場合，高い疑惑値が該当箇所に付与されないことが考えられる．もう一つの理由として，InagoFL の実装仕様が考えられる．InagoFL は仕様として実行されなかったテストケースの情報を疑惑値の計算に用いない．今回の実験において，環境の要因で実行されなかったテストケースの存在を目視で確認している．このことから，本来失敗すべきテストケースが実行されず，正しく欠陥限局が行われなかった可能性が考えられる．

6. おわりに

本研究では JavaScript におけるデバッグ作業の支援を実現するために，SBFL を行う Node.js 用モジュールであるツール InagoFL を実装した．ミューテーションツールを用いて人為的に挿入された 12 個の欠陥に対して InagoFL を適用することで欠陥検出能力を確認し，BugsJS の 10 プロジェクトに含まれる 453 個の欠陥に対して InagoFL を適用し，8 プロジェクトに含まれる 403 個の欠陥に対して正常に動作を確認し，本ツールの可搬性の確認を行った．また，可搬性の確認の際に出力された結果から，JavaScript の SBFL に対して Ample の式が有効である可能性を示した．

今後の課題として，InagoFL の対応テストフレームワークの拡張や，使用可能な SBFL 式の追加が考えられる．InagoFL は現状，Mocha テストフレームワークのみに対応しており，Jest 等の他のテストフレームワークに対して適用できない．また，多数考案されている SBFL 式のうち，InagoFL が実装できたものは 5 つのみである．加えて，疑惑値の算出の際の 0 除算に起因する疑惑値としての null の付与の解決も改善点として挙げられる．したがって，より効果的な SBFL 式の実装や，多数のプロジェクトへの適用可能性を向上させる対応テストフレームワークの増加など，本ツールにはまだ改善の余地があり，改良を加えることでより広範に用いることができる実用的なツールになる

と考えられる．

謝辞 本研究の一部は，JSPS 科研費（JP21H04877）による助成を受けた．

参考文献

[1] X. Mao, Y. Lei, Z. Dai, Y. Qi, and C. Wang, “Slice-based statistical fault localization,” Journal of Systems and Software, vol.89, pp.51–62, 2014.

[2] A. Zeller, “Isolating cause-effect chains from computer programs,” ACM SIGSOFT Software Engineering Notes, vol.27, no.6, pp.1–10, 2002.

[3] W.E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, “A survey on software fault localization,” IEEE Transactions on Software Engineering, vol.42, no.8, pp.707–740, 2016.

[4] 柗本真佑，肥後芳樹，有馬諒，谷門照斗，内藤圭吾，松尾裕幸，松本淳之介，富田裕也，華山魁生，楠本真二，“高処理効率性と高可搬性を備えた自動プログラム修正システムの開発と評価，” 情報処理学会論文誌，vol.61，no.4，pp.830–841，April 2020.

[5] R. Abreu, P. Zoeteweij, and A.J. Van Gemund, “On the accuracy of spectrum-based fault localization,” In Proceedings of Academic and industrial conference practice and research techniques-MUTATION (TAICPART-MUTATION 2007)IEEE, pp.89–98 2007.

[6] R. Abreu, P. Zoeteweij, R. Golsteijn, and A.J. Van Gemund, “A practical evaluation of spectrum-based fault localization,” Journal of Systems and Software, vol.82, no.11, pp.1780–1792, 2009.

[7] W.E. Wong, V. Debroy, R. Gao, and Y. Li, “The dstar method for effective software fault localization,” IEEE Transactions on Reliability, vol.63, no.1, pp.290–308, 2013.

[8] S. Artzi, J. Dolby, F. Tip, and M. Pistoia, “Practical fault localization for dynamic web applications,” In Proceedings of ACM/IEEE International Conference on Software Engineering, vol.1IEEE, pp.265–274 2010.

[9] Q.I. Sarhan, A. Szatmári, R. Tóth, and A. Beszédes, “Charmfl: A fault localization tool for python,” In Proceedings of IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)IEEE, pp.114–119 2021.

[10] J. Campos, A. Riboira, A. Perez, and R. Abreu, “Gzoltar: an eclipse plug-in for testing and debugging,” In Proceedings of IEEE/ACM International Conference on Automated Software Engineering, pp.378–381, 2012.

[11] H.L. Ribeiro, R.P. deAraujo, M.L. Chaim, H.A. deSouza, and F. Kon, “Jaguar: A spectrum-based fault localization tool for real-world software,” In Proceedings of IEEE International Conference on Software Testing, Verification and Validation (ICST)IEEE, pp.404–409 2018.

[12] J.A. Jones, M.J. Harrold, and J. Stasko, “Visualization of test information to assist fault localization,” In Proceedings of International Conference on Software Engineering. ICSE 2002IEEE, pp.467–477 2002.

[13] T. Janssen, R. Abreu, and A.J. vanGemund, “Zoltar: a spectrum-based fault localization tool,” In Proceedings of ESEC/FSE workshop on Software integration and evolution @ runtime, pp.23–30, ACM, New York, NY, USA, 2009.

[14] P. Gyimesi, B. Vancsics, A. Stocco, D. Mazinanian, A. Beszédes, R. Ferenc, and A. Mesbah, “Bugsjs: a benchmark of javascript bugs,” In Proceedings of IEEE Conference on Software Testing, Validation and Verification (ICST)IEEE, pp.90–101 2019.

[15] P.S. Kochhar, X. Xia, D. Lo, and S. Li, “Practitioners’ expectations on automated fault localization,” In Proceedings of International Symposium on Software Testing and Analysis, pp.165–176, 2016.

[16] B. Vancsics, A. Szatmári, and Á. Beszédes, “Relationship between the effectiveness of spectrum-based fault localization and bug-fix types in javascript programs,” In Proceedings of IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)IEEE, pp.308–319 2020.