


コード記述パターンに基づく素材コード片生成による自動プログラム修正手法 (FOSE2020 2021年発表)

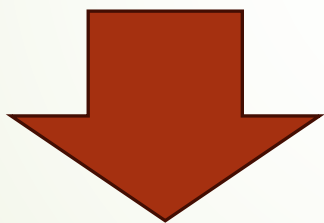


安田 和矢 伊藤 信治 中村 知倫
原田 真雄 肥後 芳樹

2172049 小池温大

研究の背景:IT業界の様子

- IT人材不足の深刻化
 - 2018年時点で約22万人のIT人材が不足!!!
 - 2030年には45万人までIT人材不足が拡大!!!(経済産業省)
- ITの需要の高まり



開発生産性を高める必要がある。

研究の背景:システム開発現場の様子

- システムの実装・テストに必要なコストの50%がデバッグ作業を占めている。^{*1}
- デバッグ作業の効率化をはかる研究が盛んに勧められている。
- その研究の中で注目されている技術として自動プログラム修正が提案されている。

概要:用語

■ 自動プログラム修正

- 一つ以上の欠陥を含むソースコードと失敗するテストケースを含むテストスイートを入力として、与えられたすべてのテストケースに成功するソースコードを出力する技術
- テストスイート テストで作成したテストケースを用いて用途や対象に応じて分類し、グループにまとめたもの

概要:自動プログラム修正手法の構成

■ 欠陥箇所特定

- ソースコードから欠陥を含む可能性(疑惑値)の高い命令を特定する処理
- Spectrum-Based Fault Localization*1、Gzoltar*2、Ochiai*3

■ パッチ生成

- 疑惑値の高い命令を書き換えることで与えられたすべてのテストケースに成功するソースコードを生成する処理
- TBar*4、TEMP-DON

1.A Survey on Software Fault Localization 2016

2.Gzoltar:An Eclipse Plug-In For Testing and Debugging 2012

3.Accuracy of Spectrum-based Fault Localization 2007

4.Revisiting Template-Based Automated Program Repair 2019

概要:TBar

■ TBarによるパッチ生成処理

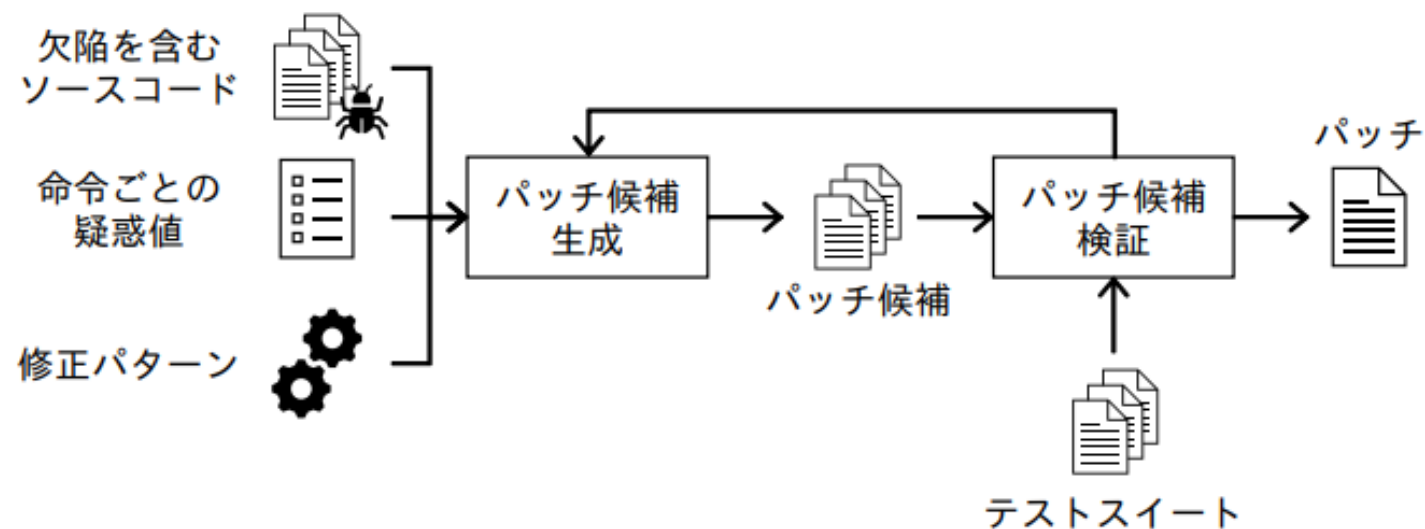
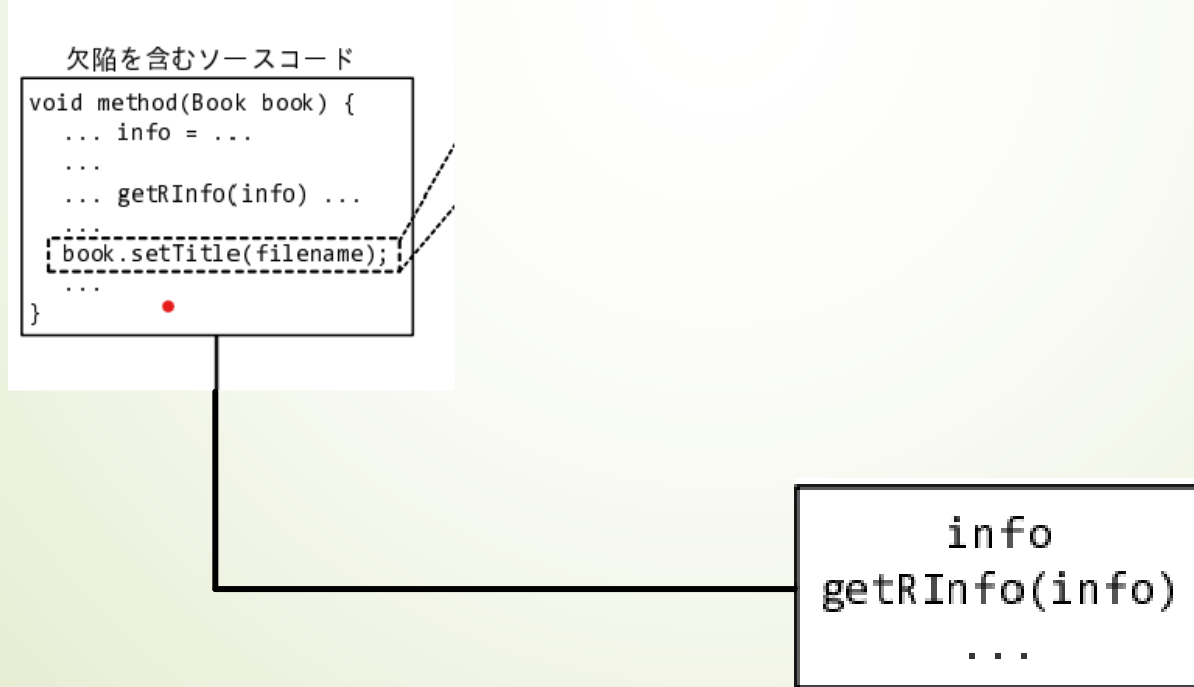


図 1 TBar のパッチ生成処理

概要:TBar

パッチ候補生成

1.ソースコード中に出現する式を素材コード片として全て取得する。



概要:TBar

■ パッチ候補生成

2. 疑惑値の高い命令を一つ選択する。

欠陥を含むソースコード

```
void method(Book book) {  
    ... info = ...  
    ...  
    ... getRInfo(info) ...  
    ...  
    book.setTitle(filename);  
    ...  
}
```

疑惑値の高い命令

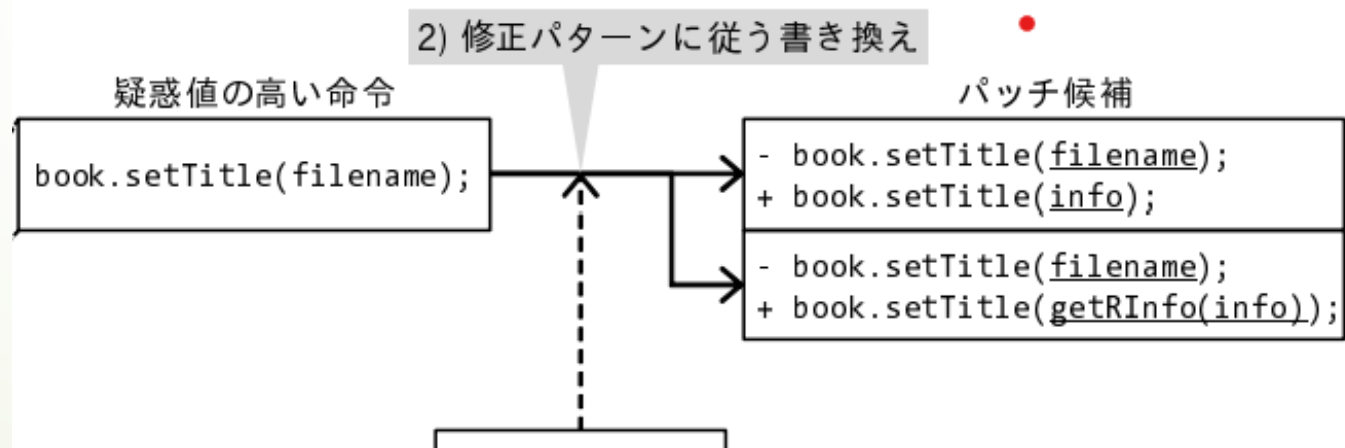
book.setTitle(filename);

概要:TBar

3.選んだ命令を事前に定義した修正パターンに従い、書き換え、複数のパッチ候補を出力する。

表 1 TBar で用いられる修正パターン[12]

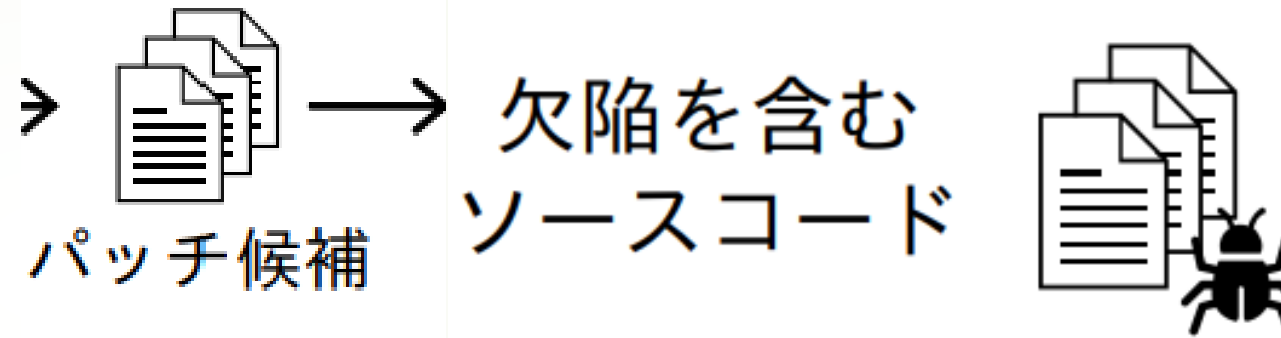
#	修正パターン名
1	Insert Cast Checker
2	Insert Null Pointer Checker
3	Insert Range Checker
4	Insert Missed Statement
5	Mutate Class Instance Creation
6	Mutate Conditional Expression
7	Mutate Data Type
8	Mutate Integer Division Operation
9	Mutate Literal Expression
10	Mutate Method Invocation Expression
11	Mutate Operators
12	Mutate Return Statement
13	Mutate Variable
14	Move Statement
15	Remove Buggy Statement



概要:TBar

パッチ候補検証

1.出力されたパッチをソースコードに適用する。





概要:TBar

パッチ候補検証

2.全てのテストケースに成功するパッチ候補があるか確認する。

概要:TBar

パッチ候補検証

パッチ候補がある場合
最終的なパッチとして出力する。

パッチ候補がない場合
パッチ候補生成からやり直す。

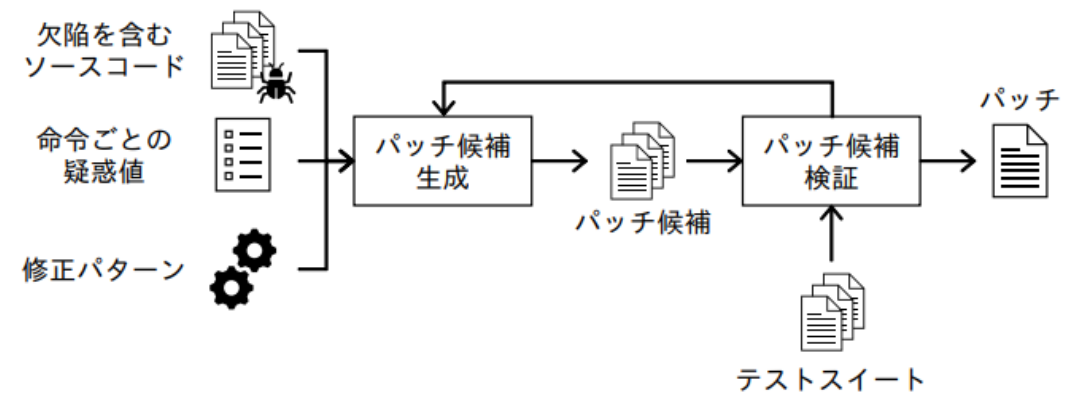


図 1 TBar のパッチ生成処理

概要:プログラム修正手法

では、必要な素材コード片がソースコード中に存在していなかったら・・・

欠陥を修正できない



提案手法

- TEMP-DON(Template-Based Program Using Donor Code Generation)

「ソースコードを記述する上でよく利用される式が存在する」

頻出式パターンに基づき生成された素材コード片によって、より多くの欠陥を修正できるように改良

TEMP-DON

■ 頻出式パターン

式の型ごとに、その型を持つ式が別の式の中でどのように利用されるかを定義したもの

```
pattern    = params, “:”, expression;  
params     = { param, “,” }, param;  
param      = classname, var; •
```

修正対象システムでよく利用される式

*param*を複数定義することができる

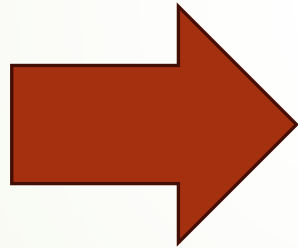
式の型

*classname*に対応する式

TEMP-DON

1.欠陥を含むソースコードから式を取得する。

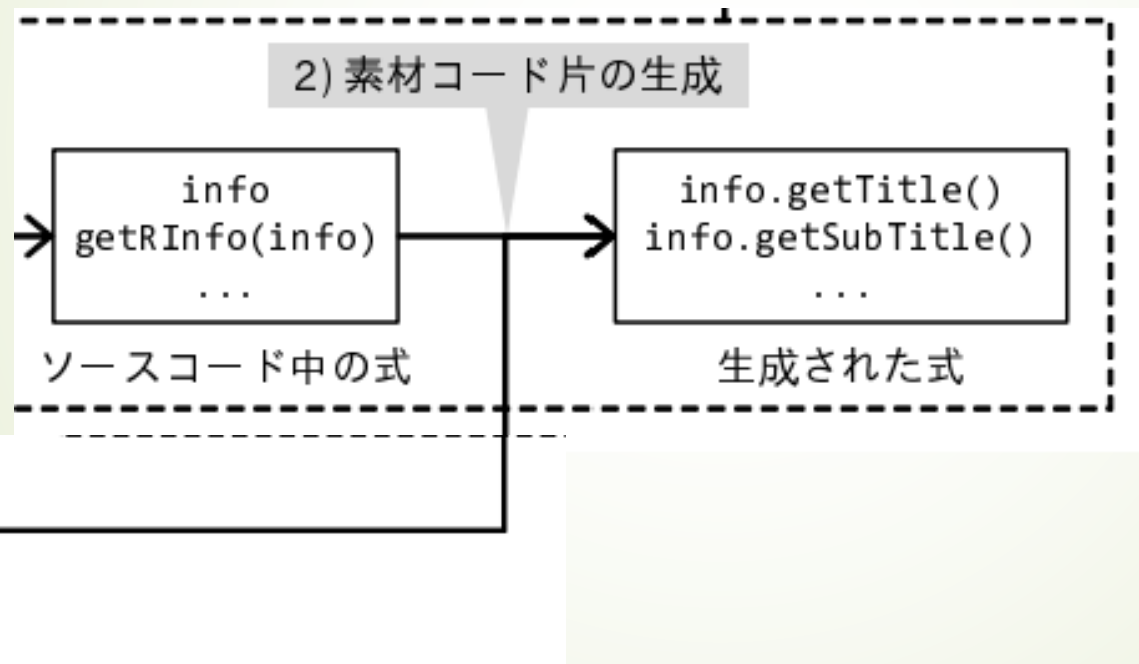
```
void method(Book book) {  
    ... info = ...  
    ...  
    ... getRInfo(info) ...  
    ...  
    book.setTitle(filename);  
    ...  
}
```



```
info  
getRInfo(info)  
...
```

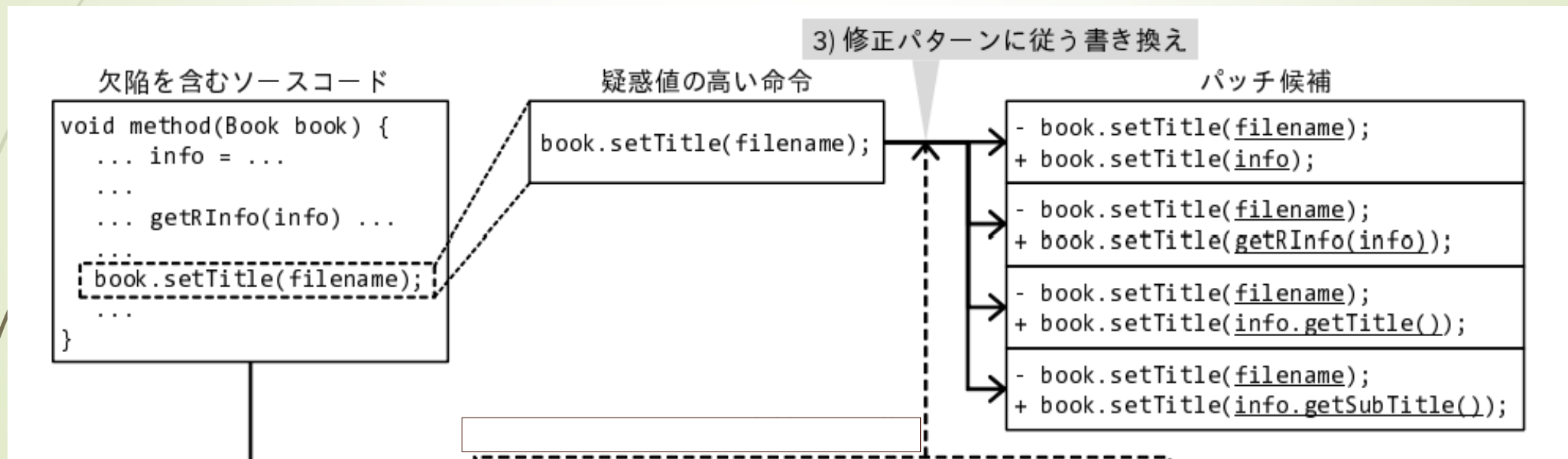
TEMP-DON

2.取得した式から頻出式パターンに従って、素材コード片を生成する。



TEMP-DON

3. 疑惑値の高い命令を修正パターンに従って、書き換える





TEMP-DON

4.既存手法のTBarと同じようにパッチをソースコードに適用し、全てのテストケースに成功するパッチ候補があるか確認する。

パッチ候補がある→最終的なパッチとして出力

パッチ候補がない→パッチ候補生成からやり直す。



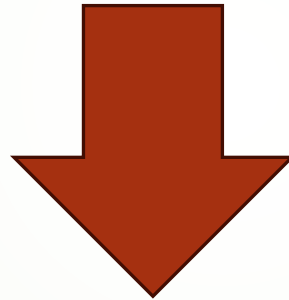
評価:概要

使用システム: TBar、TEMP-DONを実装したプロトタイプ
GZoltar(欠陥箇所特定)
Ochiai(疑惑値の算出)

評価題材 企業で開発された実システムで開発途中のある期間
に検出・修正された欠陥48件
250のクラス、約35000LoCから成る。

評価:評価方法

- 全てのテストケースに成功するパッチが生成されても実装された修正内容と等しいとは限らない



評価では開発者によって実装された修正内容と等しいパッチが出力されたパッチに含まれていることで欠陥が修正できたかどうかを確認。



評価:分類

■ 結果について以下の三つに分類する。

「パッチ未出力」

全てのテストケースに成功するパッチがない状態

「パッチ出力(開発者と等価)」

全てのテストケースに成功するパッチが出力され、修正内容と等価なパッチが含まれている。

「パッチ出力(開発者と相違)」

全てのテストケースに成功するパッチが出力されたものの、修正内容と等価なパッチが含まれていない。

評価:結果

表 2 各手法の実行結果と該当する欠陥の件数

実行結果	TEMP-DON	TBar
パッチ出力 (開発者と等価)	9	8
パッチ出力 (開発者と相違)	2	1
パッチ未出力	37	39
合計	48	48

評価:結果

表 3 各手法で実行結果が異なった欠陥

ID	Temp-Don	TBar
1	パッチ出力 (開発者と等価)	パッチ未出力
2	パッチ出力 (開発者と等価)	パッチ未出力
3	パッチ出力 (開発者と相違)	パッチ出力 (開発者と等価)

評価:結果

表 4 各手法で欠陥の修正に要した時間

	TEMP-DON	TBar
平均値	1 時間 40 分 51 秒	32 分 37 秒
中央値	24 分 54 秒	11 分 59 秒
最小値	27 秒	15 秒
最大値	7 時間 56 分 32 秒	1 時間 35 分 04 秒

考察:1


- パッチ出力できた欠陥について
 - パッチに出力できた欠陥の数
 - 既存手法(TBar):9件
 - 提案手法(TEMP-DON):11件
 - この結果から既存手法では出力できなかったパッチ2件が提案手法ではパッチを出力できたといえる。(表3のID1、ID2)
- メソッド呼び出し式を素材コード片として生成したことで、既存手法より多くの欠陥を修正できるようになった。

考察:2

- 表3のID3の欠陥について、既存手法では、開発者による修正と等価なパッチを出力されたのに対し、提案手法では、開発者による修正と相違なパッチが出力された。

表 3 各手法で実行結果が異なった欠陥

ID	Temp-Don	TBar
1	パッチ出力 (開発者と等価)	パッチ未出力
2	パッチ出力 (開発者と等価)	パッチ未出力
3	パッチ出力 (開発者と相違)	パッチ出力 (開発者と等価)



考察:2 原因

- 提案手法では、既存手法より多くのパッチ候補が生成される。
出力されたパッチは修正内容と等価か否かに関わらない。

パッチ候補が出力されてはいるが、正しくないパッチの候補の識別が不十分な場合、正しくないパッチが多く出力されると考えられる。



考察:3

- 各手法の欠陥に要した時間について
提案手法では、既存手法と比較したところ平均で約3倍の時間を要している。

CI環境での夜間実行などの使い方においては、十分実用的な修正時間であると考える。



関連研究1

GenProg

遺伝的アルゴリズムを用いてパッチを探索する。

問題点

命令単位で修正するモノであり、命令単位の操作だけで実現不可能なパッチを出力することはできない。

A Generic Method for Automatic Software Repair 2012



関連研究1

GenProg


解決策

提案手法の素材コード片生成を組み合わせることで命令単位以外の操作で実現することができる。

よって多くの欠陥を修正することができる。

関連研究2

- CAPGEN
- 命令単位での操作に加えて、式の単位での操作まで修正パターンに加えた手法。
- 操作のバリエーションが増える。
- パッチ候補の探索空間が大きくなる。
- よって効率よく探索することができる。



関連研究3


■ LSRepair

修正対象のシステムだけではなく、別システムのソースコードからも素材コード片を取得する。

膨大な量から素材コードを素材コード片を見つけている。

探索空間は増えるが、その分時間はかかる。


Live Search of Fix Ingredients for Automated Program Repair 2018



終わりに

- 今回は欠陥48件を対象にTBarでは9件、TEMP-DON11件の欠陥を修正することができた。
- 今後の課題は、提案手法TEMP-DONとTBar以外の既存手法と比較することが挙げられる。

今回の提案手法では、探索効率や探索効率、修正に要する時間について実験的に確認ができていないため、今後は評価を進めていく必要がある。



所感

自分はプログラム修正に興味がある。 今後は、プログラム修正に関する論文を比較し、効率よく欠陥を修正できる手法を使って研究を進めていきたいと考える。

プログラム修正の仕組みを知ったが、少しレベルが高く感じた。

実際に論文を読んで、説明ではわかりにくいことがあり、図から読み解くことで理解した。