

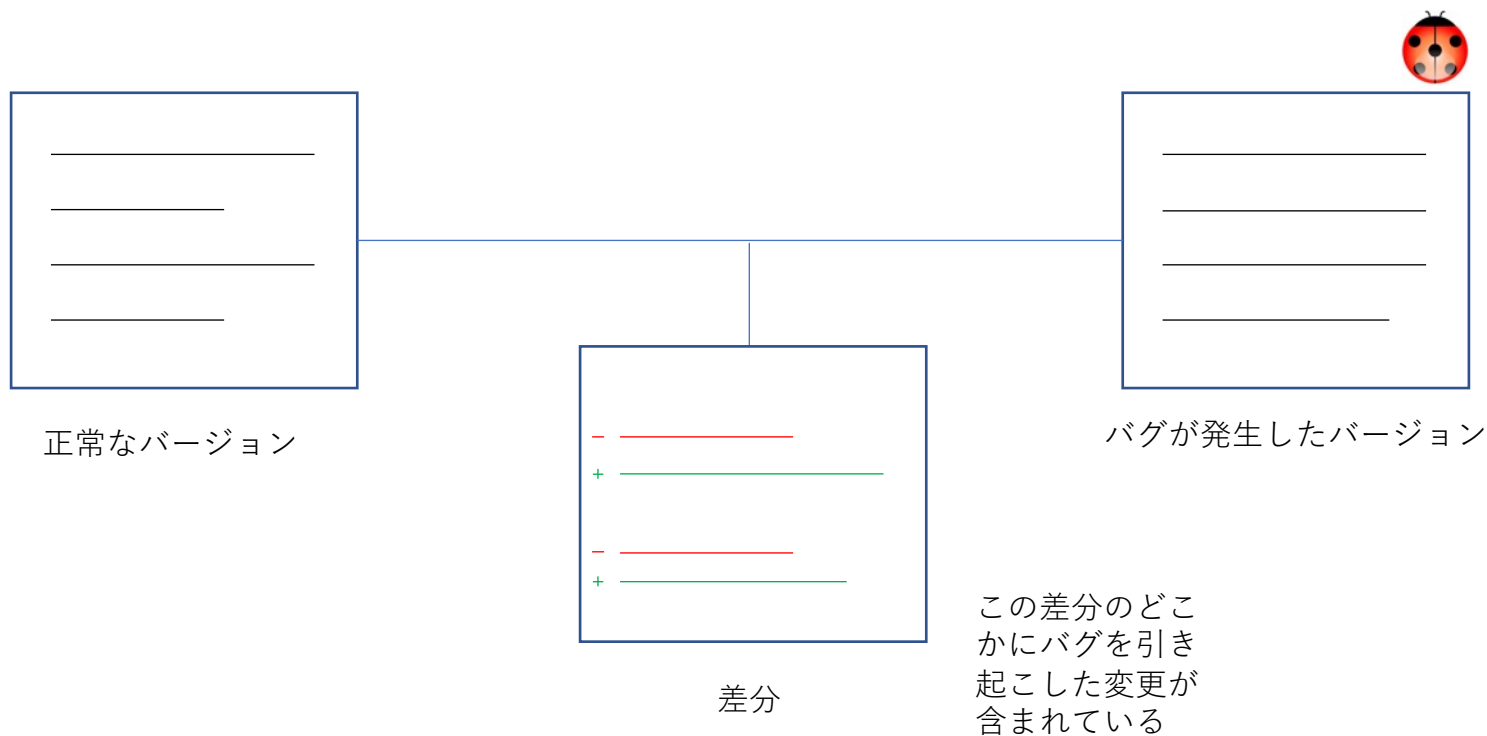
Automated Patch Extraction via Syntax- and Semantics- Aware Delta Debugging on Source Code Changes

Masatomo Hashimoto, Akira Mori, Tomonori Izumida

ESEC/FSE 2018 Proceedings of the 2018 26th ACM Joint
Meeting on European Software Engineering Conference and
Symposium on the Foundations of Software Engineering
Pages 598-609

差分デバッグ

- 正常に動作していたバージョンとバグが発生した現行のバージョンの間でコードの差分を欠陥の探索範囲とするデバッグ



差分最小化アルゴリズム[1][2]

- 得られた差分から探索すべき範囲を最小化するアルゴリズム
- 差分に含まれる変更に対してあらゆる組み合わせでテストし、テストが失敗する変更の組み合わせの中で最小のものを探索

1~8は加えられた変更
 c_i は変更の組み合わせ

Step	c_i	Configuration	test
1	c_1	1 2 3 4	✓
2	c_2	 5 6 7 8	✗
3	c_1	 5 6 . .	✓
4	c_2	 7 8	✗
5	c_1	 7 .	✗
Result		 7 .	7 is found

Table 1. Searching a single failure-inducing change

[1]Andreas Zeller. 1999. Yesterday, my program worked. Today, it does not. Why?. In Proceedings of the 7th European software engineering conference held jointly with the 7th ACM SIGSOFT international symposium on Foundations of software engineering (ESEC/FSE-7). Springer-Verlag, London, UK, UK, 253-267.

[2] Andreas Zeller and Ralf Hildebrandt. 2002. Simplifying and Isolating Failure-Inducing Input. IEEE Trans. Softw. Eng. 28, 2 (February 2002), 183-200.

問題点

- 差分が大きい場合の計算コストが大きくなる
- 変更部分の依存関係により，ビルドエラーが発生する事がある

目的

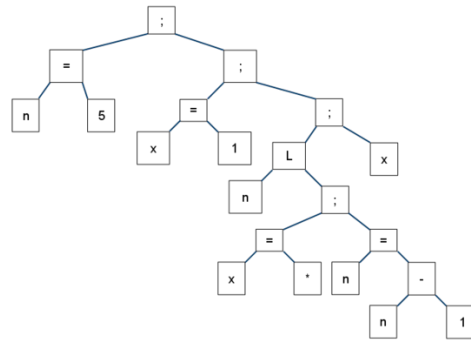
- 差分の取得の際の計算コストを削減する
- 依存関係のない変更の組み合わせを取得して差分の最小化を容易にする

アプローチ

- AST（抽象構文木）の比較による差分の取得を行うための手法の構築
- 変更の依存関係に基づくルールを用いた変更のグループ分けの手法の構築

AST

- プログラムを木構造で表現したもの
- 言語処理系でプログラムの中間表現として使われる



<http://f.hatena.ne.jp/memememomo/20100416201454>

提案手法の概要

1. 現行のバージョン（バグを含む）と1つ前のバージョン（正常）とでASTを比較する
2. 1で得られた比較結果からASTの差分を抽出する
3. 2で得られた差分から変更の依存関係のルールに基づいて変更をグループ分け

ASTの差分取得

- バージョン間でどのような操作が加えられたかを見る
 - 消去, 追加, 移動, 名前変更
- 2つのバージョン間のASTの各ノードの対応づけ
- 同じ操作をひとまとめにする

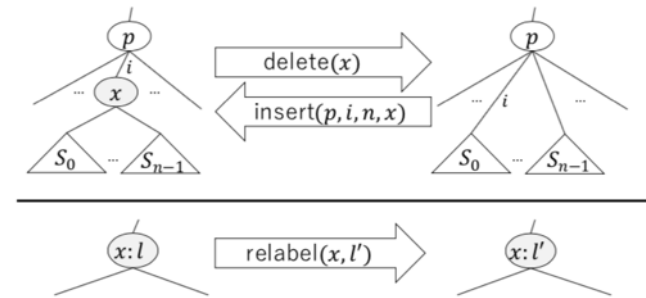


Figure 2: Tree edit operations.

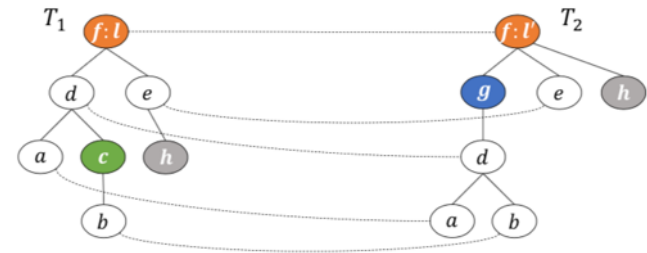


Figure 3: A tree mapping.

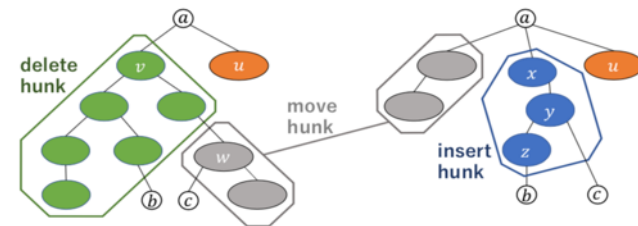


Figure 4: Tree hunks.

依存関係のルールによるグループ分け

- 構文ルール
 - 言語の構文ルールによって変更をグループ分け
 - 例) ifとelse
- リファクタリングに基づくルール
 - 各リファクタリングで決められた手順から得られる変更パターンに基づいて変更をグループ分け
- 変更に基づくルール
 - 上記以外に依存関係が構築される変更をグループ分け
 - 例) メソッド名を変更した際, 呼び出し側もメソッド名を変更する

評価実験

- 既存手法と提案手法との間で以下の性能の比較を行う
 - 差分を最小化する性能
 - 後退を抽出する性能
- 差分を最小化する性能の評価実験
 - 差分を最小化する際の、処理の規模の大きさやビルド成功率を評価
 - Defect4J[3]で収集されているバグデータを用いる
- 後退を抽出する性能の評価実験
 - 後退が発生したバージョンから原因となる変更を抽出する性能を評価
 - 本論文の筆者が収集したバグデータを用いる

差分を最小化する性能の比較

Table 1: Minimizing Reverse Bug Fixing Changes for Defects4J

B: number of bugs, P: tokens in patch, OP: tokens in common with D4J patch / tokens in patch,

NO: number of patches overlapping with no D4J patch, C: number of patch (delta) components, R: build success rate

Project	B	$P_{V_{fix}} V_{prev}$	P_{D4J}	OP_{DDP}	OP_{DDJ}	NO_{DDP}	NO_{DDJ}	C_{DDP}	C_{DDJ}	R_{DDP}	R_{DDJ}
Chart	11	1127	753	403/878	556/748	0	0	88	71	0.930 (107/115)	1.000 (125/125)
Closure	69	6057	3075	1640/3449	1189/1907	7	10	260	373	0.779 (441/566)	0.991 (541/546)
Lang	22	3384	2278	1658/2993	754/1265	1	4	64	173	0.807 (155/192)	0.986 (272/276)
Math	53	7288	4035	2383/5119	2007/3762	3	4	203	446	0.844 (356/422)	0.953 (546/573)
Mockito	30	6275	1747	1297/2280	1073/1659	0	1	225	280	0.668 (308/461)	0.965 (304/315)
Time	9	973	632	467/766	332/378	0	0	37	53	0.950 (76/80)	1.000 (89/89)
Total	194	25104	12520	7848/ 15485	5911/ 9719	11	19	877	1396	0.786 (1443/ 1836)	0.976 (1877/ 1924)

取得した差分に含まれるトークン数は提案手法の37%大きさである事がわかる

生成された変更の組み合わせは提案手法が1.6倍であるが、ビルド成功率が大きい事がわかる

後退を抽出する性能の評価実験

Table 2: Minimizing Regression Fixing Changes

P: tokens in patch, C: number of patch (delta) components, R: build success rate

Project	V_{good} (SLOC)	V_{bad} (SLOC)	$P_{V_{\text{bad}} V_{\text{good}}}$	P_{DDP}	P_{DDJ}	C_{DDP}	C_{DDJ}	R_{DDP}	R_{DDJ}
ANTLR-1543	14f05bb (15305)	aacd2a2 (14202)	9947	91	91	325	230	0.500 (20/40)	0.905 (19/21)
MATH-805	v2.0 (48495)	v2.2 (38234)	150195	728	101	3017	4320	0.149 (22/148)	0.949 (2148/2264)
HSQldb-933	r3227 (137619)	r3262 (135802)	35037	105	27	952	1317	0.127 (105/827)	1.000 (91/91)
HSQldb-1454	v2.3.2 (167638)	v2.3.3 (168563)	75907	183	75	1767	2650	0.179 (15/84)	0.676 (25/37)
JEDIT-3517	r12114 (93047)	r12710 (91556)	18462	86	2	454	634	0.555 (15/27)	1.000 (27/27)
JSOUP-920	b033535 (11146)	v1.10.3 (10565)	6792	2	2	251	451	0.230 (138/600)	0.893 (50/56)
JSOUP-926	v1.10.2 (11146)	v1.10.3 (10989)	1426	23	15	77	154	0.765 (13/17)	0.895 (17/19)
LOGBACK-1183	v1.0.13 (14905)	v1.1.0 (14700)	3072	34	21	101	165	1.000 (86/86)	1.000 (35/35)
Total	n/a (499301)	n/a (484611)	300838	1252	334	6944	9921	0.226 (414/1829)	0.946 (2412/2550)

取得した差分に含まれるトークン数は提案手法の26%の大きさである事がわかる

実行されるテストの回数は提案手法の方が大きいですが、ビルドの成功率が高い事がわかる

まとめ

- 差分最小化アルゴリズムの差分の取得をASTの比較で行う手法を構築
- 差分に含まれる変更のグループ分けを依存関係のルールに基づいて行う手法を構築
- 以上の手法により，処理すべき差分の大きさ（トークン数）を小さくする事ができたが，テスト実行回数が大きくなった

所感

- 最小化の性能の評価の部分で，実行時間がどれだけ縮んだのか知りたい
- 言語がJavaにしか対応していないので他の言語でも使えるようになって欲しい
- 読むのが難しかったけど様々な有用な技術を知る事ができた
 - 差分デバッグ，最小化アルゴリズム，Defect4Jなど