Genetic Programming for MUSEUM Fault Localisation

손정주, 유 신

본 실험에서는 실제 개발 상황에서 발생한 8 개의 결함을 statement 레벨에서 위치시키는 것을 목표로 하였다. 이 결함들은 6 가지의 multilingual 프로그램들에서 발생하였으며 각 결함들에 대한 자세한 내용은 테이블 1 에 명시되어 있다. 해당 결함들은 MUSEUM: Mutation based Debugging for Real-world Multilingual Programs 에서 성능을 측정하기 위한 타겟으로 사용되었다. MUSEUM 에서 제안한 결함 의심도 메트릭 muse 를 통해 계산된 statement 당 결함 의심도 값을 기준으로 프로그램 요소들을 정렬하였을 때 각 결함(statement)이 갖는 랭킹은 테이블 1 의 마지막 컬럼에 나와있다.

<테이블 1>

결함	타겟 프로그램	LOC (총 라인수)	# of TC (테스트 케이스 수)	결함 위치	Muse
Bug1	Azureus 3.0.4.2	340.6K	8	com.aelitis.azureus.ui.swt.views.list.ListView.java:523	1
Bug2	Sqlite-jdbc 3.7.8	4.6K	150	src/org/sqlite/Stmt.java:190	2
Bug3	Sqlite-jdbc 3.7.15	4.6K	159	src/org/sqlite/PrepStmt.java:64	1
Bug4	Java-gnome 4.0.10	64.2K	170	src/jni/bindings_java_signal.c:19586	1
Bug5	Java-gnome r-658	67.1K	184	src/bindings/org/gnome/gtk/Spell.java:67	8
Bug6	SWT 3.7.0.3	118.7L	50	org.eclipse.swt.gtk.linux.x86/temp.folder/@dot.src/org/eclipse/swt/widgets/Display.java:2602	3
Bug7	Sqlite-jdbc 3.6.0	4.9K	112	src/org/sqlite/NativeDB.c:1743	1
Bug8	SWT 4.3.0	126.6K	204	os.c:39339	1

본 실험은 MUSEUM 에서 제안한 muse score 외 여러 연구에서 효과적이라 검증 된 3 개의 SBFL 에 결함 의심도 공식-Jaccard, Ochiai, Op2-를 사용하여 생성된 결함 의심도 값들과 f2p, p2f, 그리고

각 statement 별로 생성된 뮤턴트의 수를 GP 의 피처로 사용하여 결함들을 효과적으로 위치시킬 수 있는 모델을 생성하는 것을 목표로 두었다.

<테이블 2>

피처 명	설명
MUSE	MUSE 의심도 값

Jaccard	$\frac{e_f}{\sqrt{(e_f+n_f)(e_f+e_p)}}$
Ochiai	$\frac{e_f}{\sqrt{(e_f+n_f)(e_f+e_p)}}$
Op2	$e_f - \frac{e_p}{e_p+n_p+1}$
f2p	원래 실패한 테스트 케이스가 뮤턴트에 의해 성공된 수
p2f	원래 성공한 테스트 케이스가 뮤턴트에 의해 실패된 수
num_mutants	Statement 당 생성된 뮤턴트의 수

GP 는 앞서 여러 결함 위치 추정에 관한 연구들에서 (FLUCCS, Evolving human competitive spectra-based fault localization techniques: SSBSE 2012, FLUCCS: using code and change metrics to improve fault localization: ISSTA 2017) 유효한 피처 데이터가 주어졌을 때 효과적인 위치 추정 모델을 생성할 수 있다는 것이 검증되었다. GP 를 결함 위치 추정 모델 생성에 사용하였을 때의 가장 큰 강점은 실제 유효한 정보를 포함하고 있으나 사람이 직관적으로 생각하기 힘든 영역을 탐색하여 이전에 넘어서지 못하던 성능의 한계를 뛰어넘을 수 있다는 것과 효과적인 모델을 생성하는 데 소모되는 노력과 시간을 절감할 수 있다는 것이다.

본 실험에서는 GP 의 확률적인 성격을 감안하여 30 번 반복하였으며, k 겹 (k = 4) 교차검증을 통해 오버피팅을 피하였다. 각 fold 에 속한 학습데이터와 테스트 데이터는 6 개, 2 개의 결함으로 구성되었다. 테이블 3 은 각 fold 가 어떻게 구성되었는지 보여준다.

<테이블 3>

ID	**학습 데이터**	**테스트 데이터**
0	bug1, bug2, bug3, bug4, bug5, bug6	bug7, bug8
1	bug1, bug2, bug3, bug4, bug7, bug8	bug5, bug6
2	bug1, bug2, bug5, bug6, bug7, bug8	bug3, bug4
3	bug3, bug4, bug5, bug6, bug7, bug8	Bug1, bug2

해당 연구에서 사용된 GP 는 트리 구조로 결함 위치 추정 모델을 생성하며 총 6 개의 연산자 (덧셈, 뺄셈, 나눗셈, 곱셈, 제곱근, 음수화)를 사용하였다.

MUSE score 를 포함하여 총 7 개의 피처를 학습 데이터로 사용한 GP 의 결과는 다음과 같다.

<테이블 4>

ID	**공식**	**fitness**	**ranking**
0	MUSE / (MUSE / (f2p * num_mutants)+ (f2p / Ochiai))	0.18007199	bug7: 1
			bug8: 1

1	MUSE * Op2 * (Ochiai / num_mutants + MUSE)	0.55194497	bug5: 2
			bug6: 2
2	Op2 * MUSE * (Jaccard / (num_mutants * num_mutants))	0.38168401	bug3: 1
			bug4: 1
3	Jaccard * (Ochiai / f2p) * MUSE	0.36023501	bug1: 1
			bug2: 1

테이블 4 에서 fitness score 는 실제 결함을 만나기까지 살펴본 결함이 아닌 요소(statement)들의 개수를 percentage 로 나타낸 결과이며 마지막 column 은 결함 의심도 순으로 정렬하였을 때 실제 결함들의 랭킹이다.

이를 앞 MUSEUM 결과와 비교해 보면, MUSEUM 에서 실제 결함을 정확히 위치시킨 경우 (결함 케이스 1, 3, 4, 7, 8)에 대해서는 동일하게 결함을 정확히 위치시켰으며 그 외에 MUSEUM 에서 실패한 결함 케이스 3 을 정확히 위치시켰다. 결함 케이스 2, 5 에 대해서는 실제 결함이 포함된 statement 을 랭킹 2 에 위치시켰다.

결함 2, 5 에서 결함이 랭킹 2 에 위치된 이유를 분석해 본 결과, 해당 랭킹은 두 statement 가 같은 결함 의심도 값을 가져 tie 로 인해 발생하였으며, 이 두 statement 들은 실제로 같이 실행되어 결함이 발생한 것으로 의심된다.

<테이블 5>

ID	가장 높은 결함 의심도 값을 가진 statement
bug 2	src/org/sqlite/Stmt.java:190
	src/org/sqlite/Stmt.java:197
bug 5	src/bindings/org/gnome/gtk/Spell.java:67
	generated/bindings/org/gnome/gtk/GtkSpell.java:57

Diff between 1568679 and 8e825f7

```
=== modified file 'src/main/java/org/sqlite/Stmt.java'
    public int executedUpdate(String sql){
       ...
+      changes = db.total_changes();
       ...
       db._exec(sql);
-      changes = db.changes();
+      changes = db.total_changes() - changes;
       ...
       return changes;
    }
```

<Bug 2>

Diff between rev. 658 and rev. 659

```
=== modified file 'src/bindings/org/gnome/gtk/Spell.java'
--- src/bindings/org/gnome/gtk/Spell.java 2009-06-07 11:00:30 +0000
+++ src/bindings/org/gnome/gtk/Spell.java 2009-06-07 13:54:01 +0000
    protected void release() {
-       GtkSpell.detach(this);
+   // GtkSpell.detach(this);
    }
```

<**Bug 5**. src/bindings/org/gnome/gtk/Spell.java:67: relase 함수에서 GtkSpell 의 detach 함수를 호출한다>

```
static final void detach(Spell self) {
    if (self == null) {
        throw new IllegalArgumentException("self can't be null");
    }

    synchronized (lock) {
        gtkspell_detach(pointerOf(self));
    }
}
```

<Bug 5. generated/bindings/org/gnome/gtk/GtkSpell.java:57 **>**

이와 같은 결과를 통해 본 연구는 GP 와 같은 data-driven 방식을 통해 기존의 사람이 일일이 정의하여 생성한 모델들과 비슷하거나 더 낳은 성능을 가진 모델을 생성할 수 있다는 가능성을 보여준다.