서브타입 다형성을 갖는 하위 클래스 간 유닛 테스트 교차 검사를 통한 효과적인 비일관성 오류 검출

전주영, 홍신
한동대학교 전산전자공학부
{juyoungjeon, hongshin}@handong.edu

Detecting Subtype Inconsistency Errors by Unit Test Cross-checking

요약

본 논문은 Java 프로그램에서 서브타입 다형성 구조를 갖는 하위 클래스들에 간의 동작 서브타입 비일관성 오류를 효과적으로 검출하기 위한 새로운 테스팅 기법을 제시한다. 제안하는 기법은 유닛 테스트 생성기를 통해 하위 클래스 별로 유닛 테스트 케이스를 생성한 후, 생성된 테스트 케이스를 하위 클래스 간에 교차하여 실행한 후 그 결과를 비교하여 하위 클래스 간에 테스트 케이스 결과가 일관적이지 않은 경우를 오류 발견으로 보고한다. 제안한 기법을 Apache Commons Collection에 대해 사례 연구를 수행한 결과, 실제 서브타입 비일관성 결함을 2개를 발견하였으며 제안한 기법의 효과성을 확인할 수 있었다.

1. 서론

서브타입 다형성(subtype polymorphism)은 같은 상위 클래스(superclass)를 갖는 여러 하위 클래스(subclass)들이 형태와 기능에 있어 일관성을 유지하여, 하위 클래스 객체를 상위 클래스 타입으로 참조 가능하게 하는 객체지향적 설계 방식이다. 서브타입 다형성의 형태적 일관성의 경우, 각 프로그래밍 언어의 타입 검사(예: Java Interface)를 통해 정적으로 검사가 가능하다. 반면, 동작 일관성의 경우, 정적 분석을 통한 정확한 검사가 어려우므로, 하위 클래스 간에 공유하는 동일한 이름의 메소드가 상위 클래스에서 의도대로 일관된 기능을 제공하는 지 테스팅을 통해 검사해야 한다. 만약 하위 클래스 간에 동작 일관성이 어긋날 경우, 상위 클래스 타입이 가리키는 객체가 어떠한 하위 클래스에 속하느냐에 따라 동작이 달라지는 **서브타입 비일관성 오류(subtype inconsistency error)**가 발생한다.

다음은 `AbstractList`를 상위 클래스로 갖는 두 하위 클래스 `SetUniqueList`와 `LazyList` 간에 발생하는 비일관성 오류를 탐지하는 테스트 케이스다:

```
01   test () {
02     AbstractList list = new LazyList() ;
03     LinkedList linkedlist = new LinkedList() ;
04     list.addAll(-1, linkedlist); }
```

본 테스트 케이스는 `addAll`의 첫번째 인자로 음수인 `-1`을 입력(4행)한다. 실행 결과, `AbstractList`에 정의된 메소드인 `addAll` 호출에서 `IndexOutOfBoundsException`이 발생하지만, `LazyList()`를(2행) `AbstractList`의 또다른 하위 클래스인 `SetUniqueList()`로 변경할 경우, `addAll` 호출은 `Exception`을 발생시키지 않고 정상 종료되며 실행 결과도 요구사항 명세와 다르게 잘못된 것을 확인할 수 있다[1]. 이러한 동작 비일관성은 개발자가 `AbstractList`의 `addAll`의 동작을 일관되게 예측하기 어렵게 만들어 문제를 일으키는 결함으로 볼 수 있다.

이와 같이, 같은 상위 클래스를 갖는 하위 클래스 간 동작 일관성을 검사하고, 오류 증상을 효과적으로 검출하기 위해서는 상위 클래스가 하위 클래스에 공통적으로 요구하는 일관성을 기술한 테스트 오라클(test oracle)이 필요하다. 하지만 테스트 오라클 작성은 난이도가 높고 많은 시간이 소요되며[1] 추상 클래스, 인터페이스와 같이 구체적인 소스코드가 특정되지 않은 상위 클래스에 대해서는 기존의 자동 테스트 오라클 생성 기법[2]의 적용이 어려우므로, 일반적인 프로젝트 상황에서 서브타입 다형성에 대한 면밀한 테스트 명세를 확보하는 데 한계가 큰 상황이다.

본 연구는 Java 프로그램에서 서브타입 다형성을 갖는 클래스 간의 동작 비일관성 오류를 효과적으로 검출하기 위해 (1) 하위 클래스 별로 유닛 테스트 케이스를 생성한 후[3][4], (2) 생성된 테스트 케이스를 하위 클래스 간에 교차하여 실행한 후 그 결과를 비교하여, 테스트 케이스가 비일관된 경우를 오류로 보고하는 새로운 테스팅 방법론을 제시한다. 이때, 하위클래스 메소드의 동작 중에는 상위 클래스에서 의도한 일관된 요구사항이 있음과 동시에 하위 클래스별로 의도적으로 차별화하여 작성한 동작이 있을 수 있으므로, 하위 클래스 간 테스트 케이스 실행 결과 차이를 무조건 오류로 판별할 수 없다. 이러한 이유로, 제안하는 기법은 여러 하위 클래스 간의 테스트 케이스 교차 실행 결과를 종합하는 휴리스틱을 제안하여 각 테스트 케이스마다 비일관성 오류로 추정되는 정도를 구하여 테스트 케이스와 함께 출력한다. 이를 통해, 비일관성 오류로 추정되는 정도가 높은 테스트 케이스에 대해 우선적으로 개발자 검토를 수행함으로써 실제 비일관성 오류를 효율적으로 파악할 수 있도록 하였다.

본 연구에서 제안한 기법을 자동 유닛 테스트 생성 기법인 EvoSuite [3]을 이용하여 구현하였고 Apache Commons Collection[5]에서 `AbstractCollectionDecorator`의 네 개 하위 클래스를 실험 대상으로 선택하여 사례연구를 수행하였다. 사례연구 결과, 제안한 기법은 하위 클래스 간의 서브타입 비일관성 오류 2건을 검출할 수 있었으며, 이를 분석하여, 이전에는 발견되지 않은 1건의 신규 결함을 탐지할 수 있었다.[2] 이 중 1 건의 결함은 개발자에게 보고되었고, 개발자들이 이에 대한 수정 버전을 배포하였다[6].

본 연구와 관련된 연구로 완전한 호환성이 요구되는 부모-자식 클래스 간 대체호환성(substitutablility)을 검사하기 위해 부모 클래스를 자식 클래스의 테스트 오라클로 활용하는 기법[7]이 있다. 하지만 [7]의 기법은, 본 연구에서 테스트 대상으로 하는 완전한 호환성이 요구되지 않는 하위 클래스를 대상으로는 직접 활용이 불가능하다. 본 논문에서는 완전한 호환성 대신, 클래스 별로 특이성이 의도된 하위 클래스 간의 비일관성 오류를 탐지한다는 점에서 [7]의 기법과 구별된다.

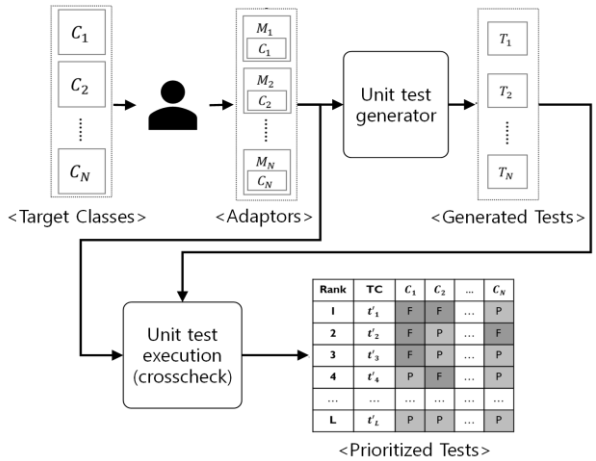
2. 유닛 테스트 교차 실행을 통한 오류 검출

2.1. 서브타입 비일관성 오류 검출 기법

§ 본 연구는 정부의 재원으로 한국연구재단의 지원을 받아 수행되었음. (NRF-2020R1C1C1013512, NRF-2017M3C4A7068179)

1 상위 클래스인 AbstractListDecorator의 Javadoc에 기술된 요구사항 명세에 따르면 이 경우 IndexOutOfBoundException이 발생하는 것이 정상 동작이다.

2 다른 1건은 상위버전인 Apache Commons 3.2에 관련 결함 보고가 존재했음.



<그림 1. 테스트 케이스 중 허위 테스트 케이스 구분 기법>

그림 1은 본 연구에서 제안하는 테스팅 기법의 전체적인 실행 과정을 나타낸다. 본 기법은 동일한 상위 클래스를 갖는 여러 개의 하위 클래스($C_1, C_2, C_3, \ldots, C_n$)들의 집합을 입력으로 받아, 최종적으로는 서브타입 비일관성 오류를 탐지하는 것으로 추정되는 테스트 케이스를 추정 정도와 함께 출력한다.

제안한 기법은 다음의 네 단계를 거쳐 수행된다:

(1) 하위 클래스 별 어댑터 클래스 작성
(2) 하위 클래스 별 테스트 케이스 자동 생성
(3) 교차 실행을 통한 비일관적 테스트 결과 탐지
(4) 비일관적 결과를 보이는 테스트 케이스에 대한 오류 추정 정도 판별

각 단계별로 구체적인 동작 방식은 다음과 같다.

1) 하위 클래스 별 어댑터 클래스 작성. 자동 유닛 테스트 케이스 생성기를 이용하여 각 하위 클래스(C_i)별로 테스트 케이스 집합(T_i)을 생성한다. 이 때, 생성된 테스트 케이스는 다른 하위 클래스에 대해서도 실행 가능하여야 하나, 주어진 하위 클래스를 그대로 이용할 경우, 다음 2가지 이유로 인해 하위 클래스 간 호환이 불가능한 테스트 케이스가 다수 생성되는 문제가 있다.

1. 상위 클래스에서 구조적으로 제한하지 않은 경우, 하위 클래스 별로 생성자 메소드(constructor)가 고유한 파라미터를 요청하는 경우가 있음.
2. 상위 클래스에서 지정하지 않은 하위 클래스의 고유한 메소드가 테스트 케이스 중에 나타날 수 있다.

이 문제를 해결하기 위해, 본 기법은 개발자와의 상호작용을 통해 주어진 하위 클래스 모두에 대한 일관된 추상 테스트 대상(abstract test target)을 작성하고 각 하위 클래스별로 어댑터(adapter)를 구현한 체계적인 방법을 구상하였다.

우선, 모든 하위 클래스가 공통적으로 갖는 메소드를 파악한 후, 각 하위 클래스 C_i에 대해 어댑터(adapter) 클래스 M_i의 초기 버전을 생성한다. 어댑터 클래스 M_i는 일반적인 생성자 메소드와 모든 하위 클래스에서 공통으로 존재하는 메소드(상위 클래스)가 선언되었으며, 하위 클래스 C_i의 객체 1개를 멤버로 갖는다. M_i의 각 메소드는 같은 이름(signature)의 C_i 메소드를 호출하고 그 결과 값을 받아서 반환하는 랩퍼(wrapper) 형태다. 이 때 각 하위 클래스 별로 생성되는 어댑터 클래스의 내용은 하위 클래스에 따라 다를 수 있으나, 모든 어댑터 클래스는 동일한 클래스 이름을 갖도록 생성한다.

이렇게 생성된 각 어댑터 클래스는 개발자에게 제공되며, 개발자는 각 하위 클래스 C_i에 특성을 고려하여 M_i가 일반적인 동작을 나타내도록 생성자 메소드를 각각 구현한다. 이 때 개발자가 판단하기에 효과적인 테스트 케이스 생성에 메소드가 추가로 필요할 경우, 이를 여러 어댑터 클래스에 일관되게 추가하여 어댑터 클래스를 완성할 수 있다.

2) 하위 클래스 별 테스트 케이스 자동 생성. 완성된 각 어댑터 클래스에 대해 자동 유닛 테스트 생성 툴을 사용하여 테스트를 생성한다. 제안한 기법은 EvoSuite [4]을 사용하여 각 어댑터 클래스에 대한 테스트 집합을 생성하도록 하였다. 이 때 생성되는 테스트 케이스는 성공(pass)하는 테스트 케이스이며, 테스트 대상에 대해 실패(예: 처리되지 않은 Exception)하는 경우는 배제하였다.

EvoSuite은 커버리지 달성 정보, 뮤테이션 테스팅 정보를 피드백으로 활용하는 탐색을 통해 테스트 대상에 대한 유닛 테스트 집합을 생성한다. 따라서, 하위 클래스에 대한 어댑터 클래스는 모두 동일한 클래스 이름, 메소드 이름을 가지나, 실행되는 코드가 실제 실행하는 하위 클래스 코드가 다르므로, 각 어댑터 클래스마다 고유한 테스트 생성이 이루어지게 된다.

3) 교차 실행을 통한 비일관적 테스트 결과 탐지. 각 어댑터 클래스 M_i에 대하여 생성된 각각의 테스트 케이스 $t_{i,k} \in T_i$를 모든 다른 어댑터 클래스 M_j에 실행시키고, 그 결과를 성공 또는 실패로 기록한다. 실패의 경우, 발생한 오류 메시지를 추가로 분석하여 오류를 일으킨 Exception의 종류와 발생 시점을 기록한다.

4) 비일관적 결과를 보이는 테스트 케이스에 대한 오류 추정 정도 판별. 하위 클래스 간의 테스트 케이스 실행 결과 차이는 서브타입 비일관성 문서 때문이거나, 혹은 특정 하위 클래스에서 고유하게 구현한 동작 때문일 수도 있다. 따라서, 본 기법은 비일관된 결과를 보이는 테스트 케이스가 얼마나 서브타입 비일관성 오류에 인한 것인지 추정하고, 오류로 추정되는 정도가 높은 순서로 테스트 케이스를 정함으로써, 발견한 비일관적 동작을 보이는 테스트 케이스에 대한 추가적인 정보를 제공한다. 하위 클래스 C_i로부터 생성된 테스트 케이스 $t_{i,k} \in T_i$의 우선 순위는 다음 두 기준을 순서대로 적용하는 휴리스틱을 통하여 정해진다:

1. 실패하는 M_j가 많을수록, 즉 C_i와 일관되지 않은 결과의 하위 클래스가 많을수록 높은 우선 순위를 부여한다.
2. $t_{i,k}$가 실패할 때 발생하는 Exception 경우(Exception 종류와 Exception 발생 위치의 조합)가 C_i에서 생성된 다른 테스트 케이스에서 드물게 발생하는 경우일수록 높은 우선 순위를 준다. 즉, 테스트 $t_{i,k}$가 M_j에 대해 발생하는 Exception 경우를 $E(M_j(t_{i,k}))$라고 할 때,
$$|\{(M_j, t_{i,l}) \mid E(M_j(t_{i,k}) = E(M_j(t_{i,l}) \text{ for } \forall M_j, \forall t_{i,l}\}|$$
값이 작을 수록 높은 우선 순위를 부여한다.

첫 번째 기준은 상위 클래스의 한 메소드를 구현한 하위 클래스 메소드들 간에 동작 불일치가 많을수록 오류일 확률이 높다는 가정을 바탕으로 정한 것이다. 두 번째 기준은 하위 클래스 간 교차적 테스트 케이스 실행에서 자주 발생하는 오류는, 해당 하위 클래스의 고유한 게 동작이 광범위하게 관찰되는 것이라고 보아, 드물게 발생하는 Exception 경우에 우선순위를 높게 주도록 한 것이다. 본 기법은 우선 순위 정보를 통해 사용자가 실제 결함에 해당하는 비일관적 동작을 가능한 빨리 검토하여 결함 탐지의 효율을 높이는 것을 목표로 한 것이다.

3. 사례 연구를 통한 평가

3.1. 실험 설정

본 논문에서 제시한 기법을 Java 프로그램에 대한 기법으로 구현하여 실제 오픈소스 프로젝트인 Apache Commons Collection 3.1 중 `AbstractCollectionDecorator`의 하위 클래스인 `FixedSizeList`, `LazyList`, `SetUniqueList`, 그리고 Commons Collection 3.2의 `GrowthList`, 이렇게 4가지 하위 클래스에 적용하는 사례 연구를 수행하였다.

<표 1. 각 클래스에서 생성된 테스트 케이스 실행 결과가 동형 클래스의 결과와 불일치하는 테스트 케이스 개수>

Target class	Num. generated TCs	Num. mismatched classes			
		3	2	1	0
FixedSizeList	55	9	0	10	36
LazyList	74	12	8	21	33
SetUniqueList	79	22	1	20	36
GrowthList	83	2	13	31	37

GrowthList의 경우, 프로젝트 개발 중 새로운 기능을 가진 하위 클래스를 추가하는 경우에 발생하는 상황을 모사하기 위하여 상위 버전(Apache Commons Collection 3.2)에서 새롭게 추가된 AbstractCollectionDecorator의 하위 클래스 중 하나를 택하여 함께 실험 대상으로 사용한 것이다.

테스트 케이스 생성 기법으로는 EvoSuite을 사용하여 테스트를 자동 생성하였다. 하나의 어댑터 클래스에 대한 테스트 생성 시 각각 30분의 시간 제한을 주었다. 이 때 4개의 어댑터 클래스의 이름과 메소드 형식이 동일하므로, 하나의 하위 클래스로부터 생성된 테스트 케이스는 다른 하위 클래스와 곧바로 링크되어 실행될 수 있다.

3.2. 실험 수행 및 결과

제안한 기법을 2 장에서 설명한 단계에 따라, 실험대상인 4 개 하위 클래스에 적용하였고, 그 결과 2 개의 서브타입 비일관성 결함을 실제로 발견할 수 있었다.

첫 번째 단계로 실험 대상 하위 클래스마다 총 27 개 메소드를 갖는 어댑터 클래스를 작성하고, 이를 EvoSuite 에 입력하여, 표 1 의 두 번째 행('# generated TC')에서 나타내고 있는 값과 같이 총 291 개의 테스트 케이스를 생성하였다. 한 하위 클래스로부터 생성된 각 테스트 케이스를 나머지 3 개 하위 클래스에 대하여 실행하여 총 873 개의 실행 결과를 구했다. 그 결과, 149 개의 테스트 케이스가 1 개 이상의 다른 하위 클래스에서 실패하여 하위 클래스 간 비일관된 결과를 보였다. 표 1 의 세 번째에서 여섯 번째 칼럼은 각 클래스에서 생성된 테스트 케이스 중 몇개의 테스트 케이스가 기존 결과와 0 부터 3 개의 하위 클래스와 불일치하는지 나타낸다.

총 149 개의 테스트 케이스는 오류 추정 기준에 따라 우순순위가 높은 순서로 정렬하였다. 정렬된 테스트 케이스를 순서대로 검토하며, 비일관성을 발견한 테스트 케이스가 실제로 오류 상황을 탐지한 것인지를 수작업으로 판별하였다. 그 결과 25 개의 테스트 케이스가 2 개의 실제 결함으로 인한 오류를 포착함을 확인할 수 있었다. 3 개 하위 클래스와 결과 불일치를 보인 45 개 테스트 케이스(표 1 의 3 열) 중 12 개 테스트 케이스가 실제 오류를 탐지한 것이며, 2 개 하위 클래스와 결과 불일치를 보인 22 개의 테스트 케이스(표 1 의 4 열) 10 개가, 1 개의 하위 클래스와 결과 불일치를 보인 81 개 테스트 케이스 중 3 개가 실제 오류를 탐지한 경우였다.

우선 순위로 정렬된 테스트 케이스 순열의 4.5 번째(동률로 인한 평균)에서 실제 서브타입 비일관성 결함 1 개를 포착할 수 있었으며, 또한 10.5 번째에서 또다른 1 개의 서브타입 비일관성 결함에 대한 테스트 케이스를 판별할 수 있었다 (동률을 같은 테스트 케이스를 고려). 그 외 나머지 테스트 케이스의 경우, 각 하위 클래스가 갖는 고유한 기능으로 인해 나타난 비일관성임을 확인할 수 있었다. 예를 들어, LazyList 의 get 메소드의 경우, 다른 하위 클래스와 달리 현재 리스트에 배정된 용량에 따라 IndexOutOfBoundsException 을 발생시키는데, LazyList 의 고유한 기능을 따른 것이다.

이러한 테스트 케이스 검토에는 Apache Commons Collection 에 대한 사전 지식이 있는 개발자 1 명의 8 시간이 소요되었다. 두 번째 우선 순위 판별 기준으로 인해 동일한 Exception 경우의 테스트 케이스가 같은 우선 순위를 갖도록 분류됨에 따라, 비슷한 이유로 비일관성을 발견한 테스트 케이스를 비교적 효율적으로 파악할 수 있었다.

3.3. 결함 사례

1) 결함 1. 본 사례 연구에서 첫 번째 발견한 결함은 addAll 메소드 호출에서 인덱스 값으로 음수 -1 을 입력할 경우, 하위 클래스에 따라서 각기 다른 결과를 보이는 결함이다(1 장의 예시는 이 결함을 간략히 표현한 것이다). 이 결함을 발견한 테스트 케이스는 SetUniqueList 의 어댑터 클래스로부터 생성되었으며, SetUniqueList 는 이 경우 Collection 을 List 마지막에 추가하고 정상종료 하지만, FixedSizeList 는 UnsupportedOperationException 을 발생시키고, LazyList 와 GrowthList 는 IndexOutOfBoundsException 발생시키는 비일관적 동작 문제를 보였다. 이 테스트 케이스는 3 개 하위 클래스 모두에서 비일관적인 결과를 보이는 경우였으며, 비일관적 결과를 보이는 총 149 개 테스트 케이스 중 4 번째 (혹은 동률로 인해 5 번째) 우선 순위에서 발견되었다. 해당 오류가 실제 결함으로 인한 것인지 파악하기 위해 SetUniqueList 의 addAll 구현을 분석한 결과, 해당 addAll 은 상위 클래스에서 기대하는 바와 다르게 인덱스로 주어진 값과 상관없이 리스트 마지막에 새로 주어진 요소를 추가하는 형태로 잘못 작성되어 있는 것을 발견하였다.

2) 결함 2. 본 사례 연구에서 발견한 또 다른 결함은 SetUniqueList 에서 생성된 다음 테스트 케이스를 통해 발견되었다:

```
01  test () {
02    LinkedList l0= new LinkedList() ;
03    AdaptorClass l1 = new AdaptorClass(l0) ;
04    l1.add(l0) ; }
```

이 테스트 케이스에서 AdaptorClass 가 LazyList 로 연결되는 경우 해당 테스트 케이스는 성공하지만 SetUniqueList 로 교차 실행될 때에는 재귀호출이 무한정 반복적으로 일어남에 따라 결국 StackOverflowException 가 발생하는 오류를 보인다. 이 테스트 케이스는 3 개의 하위 클래스 모두 비일관적인 결과를 보이는 경우였으며, 비일관적 결과를 보이는 총 149 개 테스트 케이스 중 10 번째(혹은 동률로 인해 11 번째) 우선 순위에 해당한다. SetUniqueList 를 분석한 결과, add 에서 리스트에 포함된 각 객체에 hashCode 를 호출하는 순서가 잘못되어 hashCode 가 무한정 호출되는 결함이 있음을 발견하였다. 이 결함은 개발자에게 보고되어 실제 결함 수정으로 이어졌다[6].

4. 결론 및 향후 연구

본 논문은 서브타입 다형성 구조를 갖는 하위 클래스들에 대해 테스트 케이스를 생성하고 교차 검사를 함으로 클래스의 비일관적 동작을 탐지하는 방법을 체계적인 방법론을 제시했다. 제안한 기법을 Apache Commons Collection 3.1 AbstractCollectionDecorator 의 4 개 하위 클래스에 적용하는 사례연구를 통해 2 개 실제 서브타입 비일관성 결함을 발견하였으며 제안한 기법의 효과성을 확인할 수 있었다. 향후에는 본 기법의 효용성을 증대하기 위해 완전히 자동으로 어댑터를 생성하는 방법, Randoop[4] 등의 다른 테스트 생성 기법을 사용하는 방법에 관한 연구를 하고자 한다.

참조문헌

[1] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz and S. Yoo, "The Oracle Problem in Software Testing: A Survey", IEEE Transactions on Software Engineering, vol. 41, no. 5, pp. 507-525, 2015

[2] G. Fraser and A. Zeller, "Mutation-driven Generation of Unit Tests and Oracles", ISSTA 2015

[3] G. Fraser and A. Arcuri, "1600 Faults in 100 Projects: Automatically Finding Faults While Achieving High Coverage with EvoSuite," Empirical Software Engineering, vol. 20, iss. 3, pp. 611-639, 2013

[4] C. Pacheco, S. K. Lahiri, M. D. Ernst, T. Ball, "Feedback-directed Random Test Generation", ICSE, 2007

[5] Apache Commons Collection, [Online]. Available: https://commons.apache.org/proper/commons-collections

[6] Apache Jira Issues, Collections-701, [Online]. https://issues.apache.org/jira/browse/COLLECTIONS-701

[7] M. Pradel and T. R. Gross, "Automatic Testing for Sequential and Concurrent Substitutability", ICSE, 2013