바이트코드 기반 자동 프로그램 수정 기술 동향

김성빈[O] 최준혁 남재창

한동대학교

seongbin10209@gmail.com, 21900674@handong.ac.kr, jcnam@handong.edu

Survey on Bytecode-based Automated Program Repair Technology

Seongbin Kim Junhyeok Choi Jaechang Nam

Handong Global University

요 약

소프트웨어 버그를 수정하는 작업은 많은 시간과 노력이 소요된다. 이러한 문제를 해결하기 위해 자동 프로그램 수정 기술이 등장하였다. 자동 프로그램 수정 기술에는 소스 코드를 이용하는 방식 외에도 바이트코드를 이용하는 방식이 존재한다. 바이트코드 기반 자동 프로그램 수정 기술에 대한 연구들이 소스 코드 기반 자동 프로그램 수정 기술에 비해 복잡한 버그를 수정하기 어려워 많은 관심을 받지 못했고 해당 연구 또한 비교적 활발하지 못하다. 그러나 바이트코드를 활용할 경우 컴파일 과정이 없어지기 때문에 버그 수정 속도를 유의미하게 향상할 수 있고, 자바 가상 머신(Java Virtual Machine, JVM) 기반 프로그래밍 언어에 적용할 수 있어 범용성이 좋다. 본 논문은 바이트코드 기반 APR 기술 5개를 소개함으로써 바이트코드를 기반으로 수정하는 기술의 전망을 비추고, 해당 APR들의 성능을 비교하고 장점과 한계점을 논의한다.

1. 서론

소프트웨어 버그를 수정하는 작업은 많은 시간과 노력이 소요된다. 또한 기술의 발전에 따라 소스 코드의 길이와 복잡성이 급격히 증가하여 버그 수정에 더 많은 시간과 노력이 요구된다.

이러한 문제를 해결하기 위해 자동 프로그램 수정(Automated Program Repair, APR) 기법이 등장하였다. APR은 버그를 가진 프로그램의 수정과정을 자동화하는 기술이며, 버그의 위치를 찾는 결함 위치 추정(Fault Localization, FL) 단계와 식별된 버그를 수정하는 버그 수정(Bug Repair) 단계로 구성되어 있다[1].

APR에는 소스 코드를 이용하는 방식 외에도 바이트코드를 이용하는 방식이 존재한다. 여기서 바이트코드(Bytecode)는 고급 언어로 작성된 소스 코드를 JVM이 이해할 수 있는 중간 코드로 컴파일한 것을 말한다.

바이트코드를 활용할 경우 컴파일 과정이 없어지기 때문에 버그 수정 속도를 유의미하게 향상시킬 수 있고, JVM 기반 프로그래밍 언어(예: 자바, 코틀린, 스칼라, 클로저, 그루비, 자이썬, 제이루비 등)에 적용할 수 있어 범용성이 좋다[2]. 또한 이더리움(Ethereum)이나 안드로이드(Android) 환경같이 소스 코드에 접근하기 어려운 경우에도 유용하게 활용할 수 있다. 따라서 바이트코드 기반 APR에 잠재된 연구 가치가 높다고 판단된다[2].

본 논문은 바이트코드 기반 APR 기술 소개 논문 5편을 정리하고 바이트코드를 기반 APR 기술의 전망을 비추고, 해당 APR들의 성능을 비교하며 장점과 한계점을 논의한다.

본 논문의 구성은 다음과 같다. 2장에서는 바이트코드 APR의 논문 수와 소스 코드 기반 APR의 논문 수를 비교하고 본 논문에서 다루는 APR들의 조사 방법을 소개한다. 3장에서는 바이트코드 기반 APR들을 소개한다. 4장에서는 성능과 비용을 비교한다. 5장에서는 바이트코드 기반 APR의 유용성과 한계에 대해 논의하며, 6장에서는 본 논문의 결론과 향후 연구를 제시한다.

2. 배경

최근 5년(2018~2022 년)간 *program.repair.org*에 명시된 APR 기술의 수를 조사해본 결과, 자바 언어를 기반으로 하는 APR 수는 69개, 바이트코드를 기반으로 하는 APR의 수는 6개로 추정된다. 이는 바이트코드 기반 APR이 크게 주목받고 있지 못한다는 것을 암시한다. 표1은 연도별 APR 수를 나타낸다.

표 1 2018년 ~ 2022년간 APR 논문 편수[3-77]

기술/연도	2018	2019	2020	2021	2022	총합
자바 기반 APR	13	18	7	9	22	69
바이트코드 기반 APR (소스 코드 수정)	0	1	0	0	0	1
바이트코드 기반 APR (바이트코드 수정)	1	0	1	2	1	5

다음은 바이트코드 기반 APR을 조사한 결과에 대한 설명이다. 조사 자료 선정 기준은 구글 학술 검색(Google Scholar)에서 제공하는 IEEE, ACM 소프트웨어 공학 저널 및 ICSE 학술대회 논문 중 5년 이내의(2018~2022년) 자료들로, 검색 키워드는 “bytecode automated program repair”를 사용하였다. 본 논문은 위의 자료 중에서 주제와 연관되지 않는 논문들은 제외하였고, 결함 위치 추정만을 다루거나 버그 수정에 바이트코드가 사용되지 않는 경우 또한 제외하였다.

3. 바이트코드 기반 APR 기술 소개

바이트코드 기반 APR은 바이트코드를 이용해 소스 코드를 수정하는 경우와 바이트코드를 직접 수정하는 경우로 나눌 수 있다. 소스 코드에 직접 접근할 수 있다면 소스 코드의 버그를 수정할 수 있다. 하지만 이더리움이나 안드로이드같이 소스 코드를 얻을 수 없는 경우에는 바이트코드를 수정하여 버그를 제거해야 할 필요가 있다. 논문들에서 제시된 기술들은 모두 소스 코드를 컴파일하여 얻은 바이트코드를 활용해 패치를 생성, 검증하는 과정(Generate and Validate, G & V)을 거친다[2, 78, 79, 80, 81]. 이때, 주어진 테스트 케이스를 이용해 패치를 생성하는 방법에 따라 돌연변이 기법, 템플릿 기법 등으로 구분된다.

3.1 돌연변이 기반 APR

바이트코드 돌연변이 기반 APR이란 FL로 바이트코드에서 결함을 찾아서 결함 코드를 다양한 변이 기술 및 규칙을 통해 변환하여 논리적으로 옳은 코드를 찾는 기법이다. 대표적인 바이트코드 돌연변이 기반 APR 연구로 PraPR[2]이 있다.

PraPR은 결함 위치 추정을 위해 Ochiai[82]를 이용한다. Ochiai는 통과한 테스트 케이스의 수와 실패한 테스트 케이스의 수를 사용해서 결함이 의심되는 위치를 계산하고 점수를 매기는 기법이다.

측정 후 가장 점수가 높은 위치부터 수정할 순위를 정한 후 순위에 따라 변이를 다양하게 하여 패치를 생성한다. 이때 사용된 변이들은 논문에서 주어진 다양한 변이 기술/규칙을 이용한다[2]. 변이 기술에는 연산자 변이, 조건문 변이, 메소드 교체 등 다양하다.

생성된 패치들을 테스트 케이스, 또는 오라클(Oracle) 같은 정규 규정에 따라 채택 가능한 패치를 검증한다. 채택 가능한 패치는 주어진 테스트/검증 절차 들을 모두 통과하는 경우로 제한한다[2]. 마지막으로, 개발자가 실현 가능한(Feasible) 패치를 채택하여 프로그램을 직접 수정한다.

안드로이드에서 발생하는 오류를 해결하기 위해 바이트코드 돌연변이 기반 APR이 적용된 사례는 Droix[80]이다. 안드로이드는 모바일 앱(Mobile Application) 생태계에 존재하는 프로그램 중 가장 규모가 크다. Droix는 테스트 재생기(Test Replayer), 로그 분석기(Log Analyzer), 돌연변이 생성기(Mutant Generator), 테스트 검사기(Test Checker), 코드 검사기(Code Checker), 선택기(Selector)로 구성된다.

Droix[80]는 결함의 위치를 찾기 위해 검색 기반 복구 프레임워크(Search-based Repair Framework)[83]를 이용해 실행 스택(Execution Stack)을 추적하고, 근본 문제를 해결하기 위해 8가지 유전 모델(Genetic Transformation Operators)을 적용하여 수정한다.

그림1은 Droix의 전반적인 흐름을 보여준다. Droix는 두 단계를 통해 버그를 수정한다. 첫 번째 단계에서 Droix는 계측된 APK(Android Application Package)를 생성하여 실행된 모든 콜백을 기록한다. 구성된 APK를 사용하여 Droix는 기기에서 UI 이벤트 시퀀스를 재생한다. 로그 분석기는 실행 결과 로그를 구문 분석하고, 스택 추적에서 프로그램 위치를 추출하고, 기록된 콜백을 사용하여 테스트 수준 속성을 식별한다. 그 후, 두 번째 단계에서는 변이 생성기를 통해 변이를 생성하여 기존 APK와 변이 APK를 비교한 결과를 선택자(Selector)에 보낸다. 마지막으로 평가자는 두 결과를 분석하여 속성 위반 수를 계산하고, 그 결과를 최종 고정 APK로 최고의 앱을 선택하는 선택기로 전달한다.

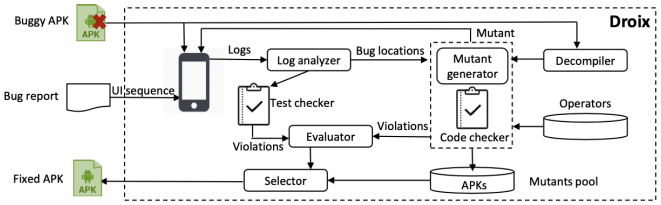

그림 1 Droix의 안드로이드 수정 프레임워크[80]

3.2 템플릿(Template)과 시맨틱(Semantic) 기반 APR

이더리움의 스마트 계약상의 버그 수정을 위해 템플릿 기반 APR 기술이 연구되어 왔다[79]. 스마트 계약상의 버그는 대부분 유사한 유형으로 발생하기 때문에 이에 강력한 템플릿 기반으로 APR 기술이 발전된 것으로 보인다[79].

스마트 계약은 보안을 위해 소스 코드를 공개하지 않기 때문에, 기존 APR은 정해진 정적 틀(탬플릿)을 이용해 주어진 바이트코드상의 결함을 찾았다. 하지만 이 템플릿들은 정적으로 하드 코딩되어 유연하지 않고, 제한적이라는 치명적인 단점이 있다[84].

이 단점을 보완하기 위해 ELYSIUM[79]이 제안되었다. ELYSIUM은 템플릿과 시맨틱을 융합하여 사용한다. ELYSIUM의 템플릿은 자리 표시자(Placeholder)로 이루어져 있어서 시맨틱을 이용해 분석한 내용과 적합하게 적용할 수 있다. 기존과 달리 문맥 정보를 추정해 템플릿으로 수정하고, 템플릿을 쉽게 확장할 수 있으며, 다른 결함 위치 추정 기술을 접목할 수 있다[79].

템플릿 기반 APR은 안드로이드의 호환성 문제에도 사용된다. 안드로이드 생태계의 심한 단편화(Fragmentation)로 인해 런타임에 충돌이 발생하거나 다른 기기에서는 잘 작동하는 앱이 특정 기기에만 설치할 수 없는 등의 심각한 호환성 문제가 발생했다. 이 문제를 해결하기 위해 호환성 문제를 자동으로 감지하고 수정하는 다양한 접근 방식이 제안되었다[81].

프로토타입 도구인 RepairDroid[81]는 사용자가 호환성 문제에 대한 수정 템플릿을 만들 수 있는 일반 앱 패치 설명 언어를 제공한다. 그런 다음 생성된 템플릿은 RepairDroid에서 활용되어 바이트코드 수준(예: 사용자가 앱을 설치하기 직전)에서 해당 문제를 자동으로 수정한다.

3.3 바이트코드 기반 기술을 활용한 패치 검증기

최근 바이트코드 기반 APR인 PraPR[2]에서 영감을 받아 기존 최신 APR로 생성한 패치를 바이트코드 레벨에서 패치 검증을 하는 패치 검증 프레임워크인 UniAPR이 연구되고 있다[78]. UniAPR은 소스 코드 기반 APR 생성한 패치와 기존 버기(Buggy) 프로젝트를 바이트코드로 컴파일한 후, HotSwap[85] 기술을 통해 융합하여 검증한다.

검증 과정은 즉석에서(On-the-fly) 패치하는 방식과 JVM을 초기화(JVM reset)한 후 패치하는 방식으로 나뉜다. 즉석에서 패치하는 방식은 전역 필드(Static Field)를 수정하는 경우 그 후의 패치들에 영향을 미칠 수 있다. 아래 그림2는 즉석 패치 검증의 부정확성을 보여준다. 해당 연구는 이 문제를 JVM을 초기화하는 방식으로 해결했다.

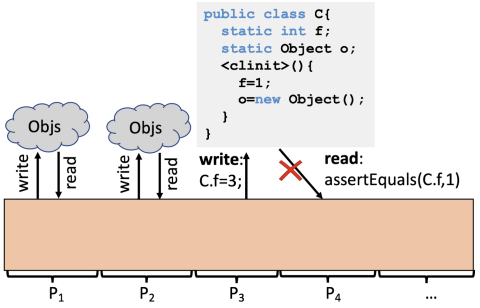

그림 2 즉석 패치 검증의 부정확성[78]

JVM 초기화 방식은 정적 오염 분석(Static Pollution Analysis), 런타임 바이트코드 변환(Runtime Bytecode Transformation), 동적 상태 초기화(Dynamic State Reset)로 이루어진다. 정적 오염 분석 단계에서 모든 클래스의 바이트코드 파일 내의 오염 될 수 있는 구간들을 식별한다. 그 후, 런타임 바이트코드 변환 단계에서 정적 클래스 이니셜라이저는 JLS(Java Language Specification)[50]에 따라 표 2에 표시된 다섯 가지 조건 중 하나라도 충족되면 클래스 초기화 이름을 바꾼다. 마지막으로 동적 상태 초기화 단계에서 각 패치 실행 후 JVM-reset 접근 방식은 상태 해시 맵(HashMap) 내의 클래스에 대한 상태를 초기화한다. 그림3은 JVM 초기화 방식의 예시이다.

표 2 클래스 초기화 상태[78]

C1	T is a class and an instance of T is created
C2	T is a class and a static method declared by T is invoked.
C3	A static field declared by T is assigned
C4	A static field declared by T is used and the field is not a constant variable
C5	T is a top level class, and an assert statement lexically nested within T is executed

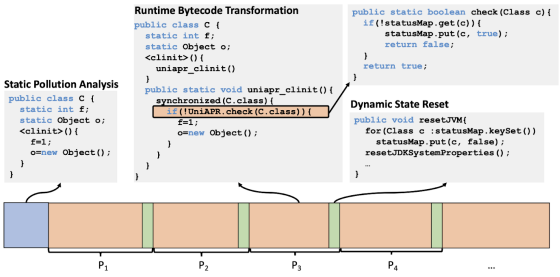

그림 3 JVM 초기화 즉석 패치 검증[78]

4. 바이트코드 기반 APR 성능 결과

앞서 소개한 논문들[2, 79, 80, 81, 78]의 각 기술에 대한 성능 실험 결과를 보고한다.

4.1 돌연변이 기반 APR 성능 결과

PraPR[2]은 Defects4J V1.2.0[86]의 버그들로 실험했을 때 최첨단 APR 기술인 SimFix[70]와 CapGen[74]보다 더 많은 버그를 수정하였다. 이러한 결과를 가져온 이유는 바이트코드 수준의 패치 생성/검증 및 실행 최적화로 인해 짧은 시간 내에 더 넓은 반경을 분석할 수 있기 때문이다[2]. 실제로 최신 자바 수정 기술들[27, 70, 71, 74]보다 10배 이상 빠르다(실행 시간 포함한 성능)[2]. PraPR은 적은 수의 후보 패치를 생성하고, 후보 패치 중 실제 수정은 original mutators들을 사용할 경우 평균 1.8위, all mutators를 사용할 경우 3.8위로 수작업이 거의 필요하지 않아서 효율적이다[2]. 또한 Defects Kotlin 버그로 PraPR에 적용한 결과 Java 버그를 수정했을 때와 비슷한 성능을 보임으로써 바이트코드 기반 APR의 확장성을 증명했다.

4.2 템플릿과 시맨틱 기반 APR 성능 결과

이더리움의 스마트 계약 보안성을 분석하고 버그를 고치기 위해 SMARTSHIELD[87], sGUARD[89], ELYSIUM[79]이 제시되었다. 그중 ELYSIUM이 가장 선두에 있다고 연구되었다.

SMARTBUGS[90] 데이터 세트를 동일한 환경으로 설정하고, 비교했다. Torres et al.[79]에서 기준으로 정한 79개의 버그를 ELYSIUM은 모두 패치에 성공했지만 SMARTSHIELD, sGUARD는 각각 45개와 33개만 패치할 수 있었다. 약 30% 더 많은 버그에 대한 패치를 생성하였다[79].

생성된 패치들의 사용 가능성에 대해 검증하기 위해 버그를 발견한 같은 방법으로 공격(스마트 계약의 보안 문제점들을 발생시키는 버그들을 수정하는 것이 목표)하였다. ELYSIUM 으로 패치된 계약들은 모든 데이터 세트에서 100% 성공적으로 방어함으로써 패치의 신뢰도를 확인하였다[79].

다음으로, 정확도를 검증하기 위해, EVMPatch[90] 및 HORUS[91] 데이터 세트에도 ELYSIUM을 적용하여 패치 후 상호작용이 패치 전과 동일하게 작동이 되는지 비교한다. EVMPatch에서는 완전히 일치하였고, HORUS 데이터 세트의 경우 96%로 기존 APR들보다 높은 결과를 보였다. SMARTSHIELD는 같은 데이터 세트에서 85% 정확도를 가졌다[79].

비용은 런타임에 발생하는 비용 위주로 계산되었다. EVMPatch과 sGUARD의 템플릿은 소스 코드로 구성되어 있어 바이트코드로 템플릿이 구성되어있는 ELYSIUM과 SMARTSHIELD에 비해 많은 초기 비용이 요구된다[79]. 이는 명령(Instruction)을 줄이기 위해 바이트코드 수준 템플릿들이 수동으로 최적화되었기 때문이다[79].

ELYSIUM과 SMARTSHIELD 중 SMARTSHIELD가 비용 측면에서 약 60~300배 더 나은 모습을 보여주지만, 이전 평가에서 볼 수 있듯이 정확성이 주요 척도일 경우 ELYSIUM이 더 좋은 성능을 보이며, 심지어 몇 개의 세트에서는 비용적으로도 나은 모습을 보인다[79].

4.3 안드로이드용 APR 성능 결과

안드로이드를 위한 APR로 버그를 해결하기 위해 Droix[80]와 호환성 문제를 해결하기 위한 RepairDroid[81]가 있다.

4.3.1 돌연변이 기반 APR 성능 결과

안드로이드의 버그를 해결하기 위해 Droix[80]가 제시되었다. Droix가 안드로이드 앱에서 발생하는 107가지 충돌의 원인을 연구한 결과, 콜백 핸들러(Callback handler) 누락(17.76%), 부적절한 리소스 처리(16%), 안드로이드 활동 및 프래그먼트 수명 주기에 대한 규칙 위반(14%)으로 인해 앱 충돌이 발생하는 것으로 조사했다.

평가된 버그 24개 중 15개의 버그의 후보 패치를 평균 30분 만에 생성했다. Droix에서 수정하지 않은 9개의 버그는 최소 10줄 이상의 편집이 필요하므로 이러한 복잡한 버그는 수정하기 어려운 것으로 나타났다[80].

4.3.2 템플릿 기반 APR 성능 결과

안드로이드 앱의 호환성 문제를 해결하기 위해 RepairDroid[81]가 제시되었다. 기존의 최신 기술인 AndroEvolve[92]에서 처리할 수 없는 모든 API를 포함하여 모든 API를 성공적으로 복구할 수 있었다. 더욱이 AndroEvolve가 여러 API 호출이 한 줄의 코드로 작성되어 있을 때 수정할 수 없었던 문제를 RepairDroid는 여러 호출이 서로 다른 줄로 구분되는 jimple 코드 수준에서 안드로이드 앱을 수정하여 해결했다.

RepairDroid는 수정 성능에서도 좋은 모습을 보인다. AndroZoo에서 1,000개의 실제 안드로이드 앱(공식 구글 플레이 스토어에 게시된 앱)을 무작위로 선택하여 테스트 데이터 세트를 구성했다. 이 중 714개의 앱에는 잠재적인 OS로 인한 호환성 문제(총 8,519개)가 포함되어 있었고, 492개의 앱은 잠재적인 장치별 문제(총 3,086개)가 있었다[81]. 7개의 앱에는 잠재적인 콜백 간 호환성 문제(총 31개)가 있었다. 확인된 모든 문제 중에서 RepairDroid는 5,932개의 OS로 인한 호환성 문제, 3,042개의 장치별 문제 및 2개의 콜백 간 호환성 문제가 아직 해결되지 않은 문제임을 확인했다[81]. 발견된 각 문제에 대해 RepairDroid는 템플릿을 적용하여 자동 수정을 수행한다. 각각 4,416개, 3,042개, 2개의 문제를 성공적으로 수정하여 각각 77.82%, 100%, 100%의 성공률을 보였다[81]. 실패 사례는 주로 RepairDroid가 템플릿에서 활용되는 변수 키워드를 기반으로 필요한 변수를 정확히 찾아내지 못하는 변수 검색 모듈에 의해 발생했다[81].

RepairDroid의 시간 성능을 조사한 결과 실제 안드로이드 앱을 분석할 때 코드 크기가 얼마나 크거나 호환성 문제가 얼마나 많은지에 관계없이 상당히 안정적임을 보였다. 이것으로 RepairDroid가 대규모 안드로이드 앱을 복구하는 데 적용하기에 적합하다는 것을 보였다[81].

4.4 바이트코드 기반 패치 검증기 성능 결과

UniAPR[78]은 최신 APR 기술 몇 개를 채택하여 실험을 수행했다. UniAPR은 31개의 자바 기반 APR 기술들을 3가지 묶음으로 분류하였을 때 각 모음 당 대표가 되는 기술을 채택하였다. 해당 연구에서 자바 HotSwap 메소드를 사용한 경우와 사용하지 않은 경우, 두 가지의 결과를 제시한다.

첫 번째 기법은 즉석 패치 검증(On-the-fly Patch Validation)이다. 분석을 위해 기존 APR의 성능과 동일한 APR을 UniAPR과 접목하였을 때 도출된 성능을 비교했다. 분석 결과 Defects4j[86]의 Math-80 환경에서 CapGen은 18,991초가 걸렸고, UniAPR은 1,590초 만에 끝냈다[78]. 이로써 즉석 패치 검증은 큰 속도 향상을 보였다. 대부분은 속도 향상을 보이지만 연구된 APR들에 따라 그리고 버그의 종류에 따라 오히려 느려지는 경우도 발생하였다. 패치를 또 다른 패치와 비교하는 작업이 추가되어 있어서 상대적으로 패치 수 적은 작은 구조에는 큰 성과를 보이지 못한다. 하지만 반대로 패치가 많고 큰 구조에는 뛰어난 성과를 보인다[78]. 마지막으로, 정확도에서는 접목된 기존 APR과 비교하였을 때 일치하지 않는 패치를 보이는 경우가 발견되었다. 이 논문을 통해 즉석 패치 검증 기술이 부정확한 결과가 나올 수 있다는 것이 발견되었다[78].

두 번째 기법은 JVM-Reset 기법이다. 이 기법은 즉석 패치 검증 방법이 실패한 패치를 모두 통과하여 채택된 APR들의 버그 패치와 패치 정확도가 일치하였다[78]. 게다가 속도 측면에서 즉석 패치 검증 기법과 유사한 성능을 가진다[78]. 그 이유는 초기화하는 빈도는 클래스 객체가 재선언이 되어 오염(변경)되는 빈도에 비례하기 때문이라고 분석되었다[78].

5. 논의

바이트코드 기반 APR을 활용하여 기존 소스 코드 기반 APR의 버그 수정 속도를 개선하고 소스 코드가 없는 환경에서도 자동으로 프로그램을 수정하는 등의 장점을 보이지만, 본 논문에서 소개한 기술들에 아직 해결해야 하거나 연구해야 하는 부분이 많이 남아 있다.

PraPR은 제한된 버그 유형에 대해서만 수정 가능한 한계가 있다[2]. 바이트코드 수준에서 단순한 변형 이외의 버그를 수정하는데 필요한 자세한 타입이나 문맥 정보를 얻기 힘들기 때문이다. 게다가 바이트코드 수준에서 복잡한 버그를 수정하기는 어렵고 복잡할 수 있다[2].

ELYSIUM[79]에도 세 가지 한계점이 있다. 먼저 평가가 제한된 스마트 계약 집합에 대해 수행되었기 때문에 결과가 이더리움 블록체인의 모든 스마트 계약에 일반화되지 않을 수 있다. 또한 평가에서는 스마트 계약이 가질 수 있는 모든 가능한 취약성의 하위 집합만 고려했으며 시스템이 모든 유형의 취약성을 감지하거나 패치하지 못할 수 있다. 이외에도 시스템이 이더리움 블록체인에서만 작동하고 다른 블록체인 플랫폼에 쉽게 적용할 수 없다. 또한 이 논문은 스마트 계약의 소유자 또는 사용자에 대한 패치 프로세스의 잠재적 영향 문제를 다루지 않는다. 시스템은 계약의 기능을 수정하지 않고 취약성이 수정되도록 보장하지만, 패치가 스마트 계약의 생태계에 영향을 미치지 않는다는 것은 보장하지 않는다. 마지막으로, 이 논문은 시스템의 확장성을 고려하지 않았기 때문에 많은 수의 스마트 계약을 처리하지 못할 수도 있다. 종합하여 ELYSIUM은 스마트 계약의 취약점을 자동으로 패치하는 유망한 접근 방식으로 보이지만 그 효과와 한계를 완전히 평가하려면 더 많은 연구와 테스트가 필요하다.

다음은 Droix의 세 가지 한계점이다[80]. 먼저 오픈 소스 앱을 수정하는 데 자주 사용되는 연산자와 안드로이드 API 설명서에서 파생한 연산자를 종합하였지만, 연산자 집합이 완전하지 않다. 또한 충돌 재현을 위해 안드로이드 에뮬레이터에 의존하므로 벤치마크의 충돌은 안드로이드 에뮬레이터에서 안정적으로 재현할 수 있는 충돌로 제한된다. 따라서 특정 설정(예: 전화 걸기)이 필요한 충돌은 재생하기가 더 어렵거나 비실용적일 수 있다[80]. 마지막으로 오픈 소스 안드로이드 앱만 조사한 연구이므로 비공개 소스 앱을 수정하는 데 사용할 수 있는지에 대한 증명이 되지 않았다[80].

RepairDroid도 세 가지 한계점을 가진다[81]. 먼저 정적 분석 결과에 영향을 줄 수 있는 반사 호출, 네이티브 코드, 멀티스레딩 기능을 인식하지 못하므로 호환성 문제의 위치가 정확하지 않을 수 있다. 또한, 현재 RepairDroid는 Soot에 정의된 가장 일반적인 표현 유형만 인식한다[81]. 드물게 [CUT] 및 [PASTE]와 같은 예약된 키워드에 비정상적인 표현이 포함된 경우, RepairDroid가 이를 인식하지 못하여 실행 가능한 명령문을 생성하지 않아 잘못된 복구가 발생한다[81]. 마지막으로 테스트 대상 기기를 찾을 수 없어 수동 확인을 통한 기기별 호환성 문제 수리의 정확성만을 평가했다는 제한점이 있다[81].

UniAPR 또한 세 가지 한계점이 있다[78]. 첫째, 패치 실행들 사이에 간섭이 발생할 수 있다. UniAPR은 정적 필드를 기본값으로 초기화하고 JDK 속성을 초기화하여 이러한 부작용을 완화했지만, 부작용이 운영 체제 또는 네트워크를 통해 전파될 수 있으므로 JVM의 메모리 내 상태를 초기화하는 것만으로는 충분하지 않을 수 있다. 둘째, HotSwap 기반 패치 유효성 검사는 필드 및 메서드를 클래스에 추가/제거 등과 같은 클래스 레이아웃 변경과 관련된 패치를 지원하지 않는다. 또한 비정적 내부 클래스, 익명 클래스 및 람다 추상화 내부에서 발생하는 패치를 지원하지 않는다. 마지막 사항은 HotSwap이 원래 정적 초기화 프로그램의 변경을 지원하지 않는다는 것이다. 흥미롭게도 JVM 초기화 접근 방식은 자연스럽게 UniAPR이 이 제한을 극복하는 데 도움이 되었다. 이제 클래스를 다시 초기화하기 위해 바이트코드 변환을 기반으로 새 초기화 프로그램을 다시 호출할 수 있기 때문이다. 따라서 JRebel[60] 및 DCEVM[59]과 같은 즉석 패치 유효성 검사를 구현하기 위한 다른 고급 동적 클래스 재정의 기술에 대한 연구가 필요하다.

다음으로 바이트코드 기반 APR을 개선하기 위한 일곱 가지 방안을 제시한다. 1) 기계 학습: 기존 바이트코드에서 학습하고 자동으로 버그를 식별하고 수정할 수 있는 새로운 기계 학습 알고리즘을 개발한다. 2) 기호 실행: 기호 실행을 사용하여 바이트코드에서 버그를 식별하는 데 사용할 수 있는 테스트 케이스를 자동으로 생성한다. 3) 하이브리드 접근 방식: 유전 알고리즘 및 제약 조건 해결과 같은 다양한 기술을 결합하여 더욱 강력하고 효과적인 APR 도구를 만든다. 4) 프로그램 분석: 버그와 취약성을 보다 정확하고 효율적으로 식별할 수 있는 바이트코드 분석을 위한 새로운 기술을 개발한다. 5) 자동화된 테스트: 바이트코드를 자동으로 테스트하는 새로운 기술을 개발하여 패치로 인해 새로운 버그가 발생하지 않도록 한다. 6) 휴먼 인 더 루프(Human-in-the-loop): 예를 들어 패치에 대한 제안을 제공하고 적용하기 전에 검토하고 승인할 수 있도록 하여 복구 프로세스에 인간 개발자를 참여시키는 방법을 개발한다. 7) 확장성: 대규모의 복잡한 소프트웨어 시스템을 처리하고 APR 도구의 확장성을 개선할 수 있는 새로운 기술을 개발한다. 이 방안들은 기존에 존재하는 소스 코드 기반 APR로부터 얻은 대략적인 방안에 불과하며 연구자들은 바이트코드 수준의 자동화된 프로그램 복구를 개선하기 위한 새로운 방법을 지속적으로 모색할 필요가 있다.

6. 결론

프로그램 개발 과정에서 큰 비중을 가지고 있는 디버깅은 점점 더 많은 시간과 비용 투자가 필요하다. 이 문제를 해결하기 위해 등장한 APR 기술은 연구가 집중되면서 발전해가고 있다[93]. 소스 코드 기반 APR들은 후보 패치를 생성하는 데 많은 시간이 필요하고, 소스 코드를 구할 수 없는 특수한 상황도 존재하므로 바이트코드 기반 APR 기술이 필요하다. 이러한 중요성에도, 아직 바이트코드 기반 APR이 소스 코드 기반에 비해 많은 관심을 받지 못하고 있다. 하지만 아직 바이트코드 기반 APR에도 과적합 문제, 바이트코드 패치들을 검증하는 방법 등 개선해야 할 문제가 많다. 따라서 바이트코드 기반 APR들에 많은 관심과 연구가 필요하다.

※이 논문은 과학기술정보통신부의 소프트웨어중심대학
지원사업(2017-0-00130)과 한국연구재단의
지원(No.2021R1F1A1063049)을 받아 수행하였음.

참고문헌

[1] Papadakis, Mike, and Yves Le Traon. "Metallaxis-FL: mutation-based fault localization." Software Testing, Verification and Reliability 25, no. 5-7, pp. 605-628, 2015.
[2] Ghanbari, Ali, Samuel Benton, and Lingming Zhang. "Practical program repair via bytecode mutation." In Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 19-30. 2019.
[3] Fonseca, Alcides, and Máximo Oliveira. "Figra: Evaluating a larger search space for Cardumen in Automatic Program Repair." In Proceedings of the Third International Workshop on Automated Program Repair, pp. 24-30. 2022.
[4] Ribeiro, Francisco, Rui Abreu, and João Saraiva. "Framing program repair as code completion." In Proceedings of the Third International Workshop on Automated Program Repair, pp. 38-45. 2022.
[5] Loriot, Benjamin, Fernanda Madeiral, and Martin Monperrus. "Styler: learning formatting conventions to repair Checkstyle violations." Empirical Software Engineering 27, no. 6, pp. 1-36, 2022
[6] Xia, Chunqiu Steven, and Lingming Zhang. "Less training, more repairing please: revisiting automated program repair via zero-shot learning." In Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 959-971. 2022.
[7] Fu, Michael, Chakkrit Tantithamthavorn, Trung Le, Van Nguyen, and Dinh Phung. "VulRepair: a T5-based automated software vulnerability repair." In Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 935-947. 2022.
[8] Li, Yi, Shaohua Wang, and Tien N. Nguyen. "DEAR: A Novel Deep Learning-based Approach for Automated Program Repair." arXiv preprint arXiv:2205.01859 (2022).
[9] Meng, Xiangxin, Xu Wang, Hongyu Zhang, Hailong Sun, and Xudong Liu. "Improving fault localization and program repair with deep semantic features and transferred knowledge." In Proceedings of the 44th International Conference on Software Engineering, pp. 1169-1180. 2022.
[10] Lee, Junhee, Seongjoon Hong, and Hakjoo Oh. "NPEX: Repairing Java Null Pointer Exceptions without Tests." (2022).
[11] Ye, He, Matias Martinez, and Martin Monperrus. "Neural program repair with execution-based backpropagation." In Proceedings of the 44th International Conference on Software Engineering, pp. 1506-1518. 2022.
[12] Gissurarson, Matthías Páll, Leonhard Applis, Annibale Panichella, Arie van Deursen, and David Sands. "PropR: property-based automatic program repair." In Proceedings of the 44th International Conference on Software Engineering, pp. 1768-1780. 2022.
[13] Li, Chengpeng, Chenguang Zhu, Wenxi Wang, and August Shi. "Repairing order-dependent flaky tests via test generation." ICSE, 2022.
[14] Zhao, Yanjie, Li Li, Kui Liu, and John Grundy. "Towards Automatically Repairing Compatibility Issues in Published Android Apps." In The 44th International Conference on Software Engineering (ICSE 2022). 2022.
[15] Benton, Samuel, Yuntong Xie, Lan Lu, Mengshi Zhang, Xia Li, and Lingming Zhang. "Towards boosting patch execution on-the-fly." In Proceedings of the 44th International Conference on Software Engineering, pp. 2165-2176. 2022.

[16] Zheng, Guolong, ThanhVu Nguyen, Simón Gutiérrez Brida, Germán Regis, Nazareno Aguirre, Marcelo F. Frias, and Hamid Bagheri. "ATR: template-based repair for alloy specifications." In Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 666-677. 2022.
[17] Yuan, Wei, Quanjun Zhang, Tieke He, Chunrong Fang, Nguyen Quoc Viet Hung, Xiaodong Hao, and Hongzhi Yin. "CIRCLE: Continual Repair across Programming Languages." arXiv preprint arXiv:2205.10956 (2022).
[18] Pinconschi, Eduard, Quang-Cuong Bui, Rui Abreu, Pedro Adão, and Riccardo Scandariato. "Maestro: a platform for benchmarking automatic program repair tools on software vulnerabilities." In Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 789-792. 2022.
[19] Ghanbari, Ali, and Andrian Marcus. "Patch Correctness Assessment in Automated Program Repair Based on the Impact of Patches on Production and Test Code." (2022).
[20] Connor, Aidan, Aaron Harris, Nathan Cooper, and Denys Poshyvanyk. "Can We Automatically Fix Bugs by Learning Edit Operations?." In 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 782-792, 2022.
[21] Lin, Bo, Shangwen Wang, Ming Wen, and Xiaoguang Mao. "Context-aware code change embedding for better patch correctness assessment." ACM Transactions on Software Engineering and Methodology (TOSEM) 31, no. 3 , pp. 1-29, 2022
[22] Xiong, Yingfei, and Bo Wang. "L2S: A framework for synthesizing the most probable program under a specification." ACM Transactions on Software Engineering and Methodology (TOSEM) 31, no. 3, pp. 1-45, 2022
[23] Tian, Haoye, Yinghua Li, Weiguo Pian, Abdoul Kader Kabore, Kui Liu, Andrew Habib, Jacques Klein, and Tegawendé F. Bissyandé. "Predicting Patch Correctness Based on the Similarity of Failing Test Cases." ACM Transactions on Software Engineering and Methodology (2022).
[24] Ye, He, Jian Gu, Matias Martinez, Thomas Durieux, and Martin Monperrus. "Automated classification of overfitting patches with statically extracted code features." IEEE Transactions on Software Engineering (2021).
[25] Kechagia, Maria, Sergey Mechtaev, Federica Sarro, and Mark Harman. "Evaluating automatic program repair capabilities to repair API misuses." IEEE Transactions on Software Engineering (2021).
[26] Chen, Zimin, Steve Kommrusch, Michele Tufano, Louis-Noël Pouchet, Denys Poshyvanyk, and Martin Monperrus. "Sequencer: Sequence-to-sequence learning for end-to-end program repair." IEEE Transactions on Software Engineering 47, no. 9, pp. 1943-1959, 2019.
[27] Chen, Liushan, Yu Pei, and Carlo A. Furia. "Contract-based program repair without the contracts: An extended study." IEEE Transactions on Software Engineering 47, no. 12, pp. 2841-2857, 2020
[28] Coker, Zack, Joshua Sunshine, and Claire Le Goues. "Framefix: Automatically repairing statically-detected directive violations in framework applications." In 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 201-212, 2021.
[29] Shariffdeen, Ridwan, Yannic Noller, Lars Grunske, and Abhik Roychoudhury. "Concolic program repair." In Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, pp. 390-405. 2021.
[30] Xu, Tongtong, Minxue Pan, Yu Pei, Guiyin Li, Xia Zeng, Tian Zhang, Yuetang Deng, and Xuandong Li. "Guider: Gui structure and vision co-guided test script repair for android apps." In Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 191-203. 2021.
[31] Noda, Kunihiro, Haruki Yokoyama, and Shinji Kikuchi. "Sirius: Static Program Repair with Dependence Graph-Based Systematic Edit Patterns." In 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 437-447, 2021.
[32] Liang, Jingjing, Ruyi Ji, Jiajun Jiang, Shurui Zhou, Yiling Lou, Yingfei Xiong, and Gang Huang. "Interactive Patch Filtering as Debugging Aid." In 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 239-250, 2021.
[33] Jiang, Nan, Thibaud Lutellier, and Lin Tan. "Cure: Code-aware neural machine translation for automatic program repair." In 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp. 1161-1173, 2021.
[34] Wong, Chu-Pan, Priscila Santiesteban, Christian Kästner, and Claire Le Goues. "VarFix: balancing edit expressiveness and search effectiveness in automated program repair." In Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 354-366. 2021.
[35] Zhu, Qihao, Zeyu Sun, Yuan-an Xiao, Wenjie Zhang, Kang Yuan, Yingfei Xiong, and Lu Zhang. "A syntax-guided edit decoder for neural program repair." In Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 341-353. 2021.
[36] Ye, He, Matias Martinez, and Martin Monperrus. "Automated patch assessment for program repair at scale." Empirical Software Engineering 26, no. 2 , pp. 1-38, 2021
[37] Tian, Haoye, Kui Liu, Abdoul Kader Kaboré, Anil Koyuncu, Li Li, Jacques Klein, and Tegawendé F. Bissyandé. "Evaluating representation learning of code changes for predicting patch correctness in program repair." In 2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 981-992, 2020.
[38] Koyuncu, Anil, Kui Liu, Tegawendé F. Bissyandé, Dongsun Kim, Jacques Klein, Martin Monperrus, and Yves Le Traon. "Fixminer: Mining relevant fix patterns for automated program repair." Empirical Software Engineering 25, no. 3, pp. 1980-2024, 2020.
[39] Ghanbari, Ali, and Andrian Marcus. "PRF: a framework for building automatic program repair prototypes for JVM-based languages." In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 1626-1629. 2020.
[40] Lutellier, Thibaud, Hung Viet Pham, Lawrence Pang, Yitong Li, Moshi Wei, and Lin Tan. "Coconut: combining context-aware neural translation models using ensemble for program repair." In Proceedings of the 29th ACM SIGSOFT international symposium on software testing and analysis, pp. 101-114. 2020.
[41] Yang, Geunseok, Kyeongsic Min, and Byungjeong Lee. "Applying deep learning algorithm to automatic bug localization and repair." In Proceedings of the 35th Annual ACM symposium on applied computing, pp. 1634-1641. 2020.
[42] Yu, Xiao Liang, Omar Al-Bataineh, David Lo, and Abhik Roychoudhury. "Smart contract repair." ACM Transactions on Software Engineering and Methodology (TOSEM) 29, no. 4, pp. 1-32, 2020
[43] Yuan, Yuan, and Wolfgang Banzhaf. "Toward better evolutionary program repair: An integrated approach." ACM Transactions on Software Engineering and Methodology (TOSEM) 29, no. 1, pp. 1-53, 2020
[44] Yuan, Yuan, and Wolfgang Banzhaf. "Arja: Automated repair of java programs via multi-objective genetic programming." IEEE Transactions on software engineering 46, no. 10, pp.1040-1067, 2018
[45] Tian, Haoye, Kui Liu, Abdoul Kader Kaboré, Anil Koyuncu, Li Li, Jacques Klein, and Tegawendé F. Bissyandé. "Evaluating representation learning of code changes for predicting patch correctness in program repair." In 2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 981-992, 2020.
[46] Jiang, Jiajun, Luyao Ren, Yingfei Xiong, and Lingming Zhang. "Inferring program transformations from singular examples via big code." In 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 255-266, 2019.
[47] Ghanbari, Ali, Samuel Benton, and Lingming Zhang. "Practical program repair via bytecode mutation." In Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 19-30. 2019.
[48] Kim, Jindae, and Sunghun Kim. "Automatic patch generation with context-based change application." Empirical Software Engineering 24, no. 6, pp. 4071-4106, 2019.
[49] Mesbah, Ali, Andrew Rice, Emily Johnston, Nick Glorioso, and Edward Aftandilian. "Deepdelta: learning to repair compilation errors." In Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 925-936. 2019.
[50] An, Gabin, Aymeric Blot, Justyna Petke, and Shin Yoo. "PyGGI 2.0: Language independent genetic improvement framework." In Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 1100-1104. 2019.
[51] Koyuncu, Anil, Kui Liu, Tegawendé F. Bissyandé, Dongsun Kim, Martin Monperrus, Jacques Klein, and Yves Le Traon. "iFixR: Bug report driven program repair." In Proceedings of the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering, pp. 314-325. 2019.
[52] Xin, Qi, and Steven Reiss. "Better code search and reuse for better program repair." In 2019 IEEE/ACM International Workshop on Genetic Improvement (GI), pp. 10-17, 2019.
[53] Saha, Seemanta. "Harnessing evolution for multi-hunk program repair." In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), pp. 13-24, 2019.
[54] Ghorbani, Negar, Joshua Garcia, and Sam Malek. "Detection and repair of architectural inconsistencies in java." In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), pp. 560-571, 2019.
[55] Xu, Xuezheng, Yulei Sui, Hua Yan, and Jingling Xue. "VFix: value-flow-guided precise program repair for null pointer dereferences." In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), pp. 512-523, 2019.
[56] Martinez, Matias, and Martin Monperrus. "Astor: Exploring the design space of generate-and-validate program repair beyond GenProg." Journal of Systems and Software 151, pp. 65-80, 2019.
[57] Li, Guangpu, Haopeng Liu, Xianglan Chen, Haryadi S. Gunawi, and Shan Lu. "Dfix: automatically fixing timing bugs in distributed systems." In Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 994-1009. 2019.

[58] Pan, Rong, Qinheping Hu, Gaowei Xu, and Loris D'Antoni. "Automatic repair of regular expressions." Proceedings of the ACM on Programming Languages 3, no. OOPSLA, pp. 1-29, 2019.
[59] Bader, Johannes, Andrew Scott, Michael Pradel, and Satish Chandra. "Getafix: Learning to fix bugs automatically." Proceedings of the ACM on Programming Languages 3, no. OOPSLA, pp. 1-27, 2019.
[60]] Liu, Kui, Anil Koyuncu, Dongsun Kim, and Tegawendé F. Bissyandé. "Avatar: Fixing semantic bugs with fix patterns of static analysis violations." In 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 1-12, 2019.
[61] White, Martin, Michele Tufano, Matias Martinez, Martin Monperrus, and Denys Poshyvanyk. "Sorting and transforming program repair ingredients via deep learning code similarities." In 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 479-490, 2019.
[62] Liu, Kui, Anil Koyuncu, Dongsun Kim, and Tegawendé F. Bissyandé. "TBar: Revisiting template-based automated program repair." In Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 31-42. 2019.
[63] Monperrus, Martin, Simon Urli, Thomas Durieux, Matias Martinez, Benoit Baudry, and Lionel Seinturier. "Repairnator patches programs automatically." Ubiquity 2019, no. July, pp. 1-12, 2019.
[64] Banerjee, Abhijeet, Lee Kee Chong, Clément Ballabriga, and Abhik Roychoudhury. "Energypatch: Repairing resource leaks to improve energy-efficiency of android apps." IEEE Transactions on Software Engineering 44, no. 5, pp. 470-490, 2017
[65] Martinez, Matias, and Martin Monperrus. "Ultra-large repair search space with automatically mined templates: The cardumen mode of astor." In International Symposium on Search Based Software Engineering, pp. 65-86, 2018.
[66] Liu, Xuliang, and Hao Zhong. "Mining stackoverflow for program repair." In 2018 IEEE 25th international conference on software analysis, evolution and reengineering (SANER), pp. 118-129, 2018.
[67] Macho, Christian, Shane McIntosh, and Martin Pinzger. "Automatically repairing dependency-related build breakage." In 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 106-117, 2018.
[68] Lee, Junho, Dowon Song, Sunbeom So, and Hakjoo Oh. "Automatic diagnosis and correction of logical errors for functional programming assignments." Proceedings of the ACM on Programming Languages 2, no. OOPSLA, pp. 1-30, 2018
[69] Wang, Ke, Rishabh Singh, and Zhendong Su. "Search, align, and repair: data-driven feedback generation for introductory programming exercises." In Proceedings of the 39th ACM SIGPLAN conference on programming language design and implementation, pp. 481-495. 2018.
[70] Jiang, Jiajun, Yingfei Xiong, Hongyu Zhang, Qing Gao, and Xiangqun Chen. "Shaping program repair space with existing patches and similar code." In Proceedings of the 27th ACM SIGSOFT international symposium on software testing and analysis, pp. 298-309. 2018.
[71] Hua, Jinru, Mengshi Zhang, Kaiyuan Wang, and Sarfraz Khurshid. "Towards practical program repair with on-demand candidate generation." In Proceedings of the 40th international conference on software engineering, pp. 12-23. 2018.
[72] van Tonder, Rijnard, and Claire Le Goues. "Static automated program repair for heap properties." In Proceedings of the 40th International Conference on Software Engineering, pp. 151-162. 2018.
[73] Tan, Shin Hwei, Zhen Dong, Xiang Gao, and Abhik Roychoudhury. "Repairing crashes in android apps." In 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), pp. 187-198, 2018.
[74] Wen, Ming, Junjie Chen, Rongxin Wu, Dan Hao, and Shing-Chi Cheung. "Context-aware patch generation for better automated program repair." In 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), pp. 1-11, 2018.
[75] Lin, Huarui, Zan Wang, Shuang Liu, Jun Sun, Dongdi Zhang, and Guangning Wei. "Pfix: fixing concurrency bugs based on memory access patterns." In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, pp. 589-600. 2018.
[76] Higo, Yoshiki, Shinsuke Matsumoto, Ryo Arima, Akito Tanikado, Keigo Naitou, Junnosuke Matsumoto, Yuya Tomida, and Shinji Kusumoto. "kGenProg: A High-performance, High-extensibility and High-portability APR System." In 2018 25th Asia-Pacific Software Engineering Conference (APSEC), pp. 697-698, 2018.
[77] Liu, Kui, Anil Koyuncu, Kisub Kim, Dongsun Kim, and Tegawendé F. Bissyandé. "LSRepair: Live search of fix ingredients for automated program repair." In 2018 25th Asia-Pacific Software Engineering Conference (APSEC), pp. 658-662, 2018.

[78] Chen, Lingchao, Yicheng Ouyang, and Lingming Zhang. "Fast and precise on-the-fly patch validation for all." In 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp. 1123-1134, 2021.
[79] Ferreira Torres, Christof, Hugo Jonker, and Radu State. "Elysium: Context-Aware Bytecode-Level Patching to Automatically Heal Vulnerable Smart Contracts." In Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses, pp. 115-128. 2022.
[80] Tan, Shin Hwei, Zhen Dong, Xiang Gao, and Abhik Roychoudhury. "Repairing crashes in android apps." In 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), pp. 187-198, 2018.
[81] Zhao, Yanjie, Li Li, Kui Liu, and John Grundy. "Towards Automatically Repairing Compatibility Issues in Published Android Apps." In The 44th International Conference on Software Engineering (ICSE 2022). 2022.
[82] Abreu, Rui, Peter Zoeteweij, and Arjan JC Van Gemund. "On the accuracy of spectrum-based fault localization." In Testing: Academic and industrial conference practice and research techniques-MUTATION (TAICPART-MUTATION 2007), pp. 89-98, 2007.
[83] Tan, Shin Hwei, Hiroaki Yoshida, Mukul R. Prasad, and Abhik Roychoudhury. "Anti-patterns in search-based program repair." In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 727-738. 2016.
[84] Ma, Siqi, David Lo, Teng Li, and Robert H. Deng. "Cdrep: Automatic repair of cryptographic misuses in android applications." In Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, pp. 711-722. 2016.
[85] Oracle Corporation. 2020. Java Instrumentation API. https://bit.ly/3czmzFV Accessed: May, 2020.
[86] Just, René, Darioush Jalali, and Michael D. Ernst. "Defects4J: A database of existing faults to enable controlled testing studies for Java programs." In Proceedings of the 2014 International Symposium on Software Testing and Analysis, pp. 437-440. 2014.
[87] Zhang, Yuyao, Siqi Ma, Juanru Li, Kailai Li, Surya Nepal, and Dawu Gu. "Smartshield: Automatic smart contract protection made easy." In 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 23-34, 2020.
[88] Nguyen, Tai D., Long H. Pham, and Jun Sun. "SGUARD: towards fixing vulnerable smart contracts automatically." In 2021 IEEE Symposium on Security and Privacy (SP), pp. 1215-1229, 2021.
[89] Ferreira, Joã̃o F., Pedro Cruz, Thomas Durieux, and Rui Abreu. "Smartbugs: A framework to analyze solidity smart contracts." In Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering, pp. 1349-1352. 2020.
[90] UDE Secure Software System Research Group. "EVMPatch Evaluation Data. Dec. 2020." URL: https://github. com/uni-due-syssec/evmpatch-eval-data.
[91] Ferreira Torres, Christof, Antonio Ken Iannillo, and Arthur Gervais. "The eye of horus: Spotting and analyzing attacks on ethereum smart contracts." In International Conference on Financial Cryptography and Data Security, pp. 33-52. Springer, Berlin, Heidelberg, 2021.
[92] Haryono, Stefanus A., Ferdian Thung, David Lo, Lingxiao Jiang, Julia Lawall, Hong Jin Kang, Lucas Serrano, and Gilles Muller. "AndroEvolve: Automated Android API update with data flow analysis and variable denormalization." Empirical Software Engineering 27, no. 3, pp. 1-31, 2022
[93] Gazzola, Luca, Daniela Micucci, and Leonardo Mariani. "Automatic software repair: A survey." In Proceedings of the 40th International Conference on Software Engineering, pp. 1219-1219. 2018.