스테이트먼트 유형 정보를 활용한 옳은 패치 생성률 증대 기법

허진석○, 정호현*, 이은석
성균관대학교 전자전기공학부, 성균관대학교 전자전기컴퓨터공학과*
{mrhjs225, jeonghh89, leees}@skku.edu

A technique to increase rate of correct patches generation using statement type information

Jinseok Heo○, Hohyeon Jeong*, Eunseok Lee
Department of Electronic and Electrical Engineering, Sungkyunkwan University
Department of Electrical and Computer Engineering, Sungkyunkwan University*

요 약

최근 디버깅(Debugging) 작업의 효율을 높이기 위해 자동으로 패치를 생성하는 프로그램 자동 수정 기법(APR, Automated Program Repair)이 연구되고 있다. APR의 한 종류인 탐사기반(Search-based) APR의 경우 탐사 영역(Search space)에 따라 옳은 패치(Correct patch)의 생성률이 영향을 받는다. 본 논문에서는 옳은 패치의 생성률 향상을 위해 SeeType을 제안한다. SeeType은 과거의 패치로부터 AST(AST, Abstract Syntax Tree)와 AST 노드(Node) 유형이 담긴 템플릿(template)을 생성한다. 그리고 식별된 결함 위치의 스테이트먼트 유형(Statement type)을 분석하고, 사람이 작성한 패치에서 패치 전 결함 위치의 스테이트먼트 유형이 분석 결과와 일치하는 템플릿을 선별하여 수정 템플릿을 적용한다. 이를 통해서 사람이 작성한 패치와 유사한 패치를 자동 생성 할 수 있으며, 따라서 옳은 패치 생성률 향상 및 디버깅 효율 향상을 기대할 수 있다.

1. 서 론

소프트웨어 유지보수는 개발자의 수동적인 작업으로 인해 비용이 많이 소모되는 작업이다. 이러한 부분을 개선하는 방법으로 결함 추적부터 패치 생성까지 자동으로 진행되는 APR이 활발히 연구되고 있다[1, 2].

그러나 APR 기법의 하나인 탐사 기반 APR 기법이 생성한 패치의 질이 떨어진다는 지적이 존재한다[3, 4]. 탐사 기반 APR의 성능은 탐사 영역에 기반하므로[5], 이러한 점을 극복하기 위해서는 탐사 영역을 잘 구성해야 한다. 또한 [6]은 단순히 탐사 영역을 늘릴 경우 옳은 패치뿐만 아니라 테스트 케이스는 통과하나 개발자가 수용할 수 없는 패치인 그럴듯한 패치(plausible patch)가 증가한다고 밝혔다. 따라서 옳은 패치를 생성하기 위해서는 옳은 패치가 존재하는 질 높은 탐사 영역만을 추출하여 이를 기반으로 패치를 생성해야만 한다.

기존부터 옳은 패치와 관련하여 탐사영역을 어떻게 구성할지에 대한 연구가 진행되고 있다. 연구 [7]은 스테이트먼트와 같은 큰 단위(coarse-grained granularity)의 정보로는 많은 버그에 대해 옳은 패치를 생성하기 어렵고, AST 노드와 같은 작은 단위(fine-grained granularity)의 정보로는 옳은 패치를 생성할 수 있으나 효율적이지 못한다고 밝혔다. 이를 기반으로 본 논문에서는 옳은 패치의 생성률 향상을 목표로 기존의 연구[8, 9, 10]에서 사용하지 않은 작은 단위 정보인 사람이 작성한 패치의 스테이트먼트 AST 노드 유형 정보를 활용한 SeeType (**S**earch spac**e** r**e**duction using Statement **type**) 기법을 제안한다.

SeeType은 크게 두 단계의 처리 과정을 가진다: 1) 사람이 작성한 패치 정보 수집 및 분석을 통한 패치 생성용 템플릿 자동 작성, 2) 식별된 결함 위치의 스테이트먼트 유형에 따른 패치 생성 템플릿 적용 및 평가.

패치 생성에 사용될 템플릿을 생성하는 과정에서는 먼저 오픈 소스 소프트웨어 저장소의 프로젝트에서 사람이 작성한 패치를 수집한다. 해당 패치들의 패치 전, 후 스테이트먼트에 대해 AST 노드 유형 분석을 진행한 후, 패치 전, 후의 AST 노드 유형 쌍과 AST를 템플릿으로 취급하여 추출한다.

이후 버그가 발생한 프로젝트에 대해 APR 과정을 진행한다. 먼저, 버기 코드(Buggy code)와 테스트 케이스를 통해 결함 추적을 한다. 결함 추적 결과로 나온 의심스러운 스테이트먼트의 AST 노드 유형 분석을 진행한다. 미리 만들어 놓은 템플릿 중 의심스러운 스테이트먼트의 AST 노드 유형과 패치 전 AST 노드 유형이 같은 템플릿만 추출한다. 추출한 템플릿을 통해 후보 패치를 생성하고 평가 과정을 거쳐 유효한 패치(Valid patch)를 생성한다.

SeeType에서의 탐사영역은 패치 생성을 위해 사용되는 모든 템플릿을 의미한다. 패치 생성 시 모든 템플릿에 대하여 탐색하는 것이 아닌, 결함 추적의

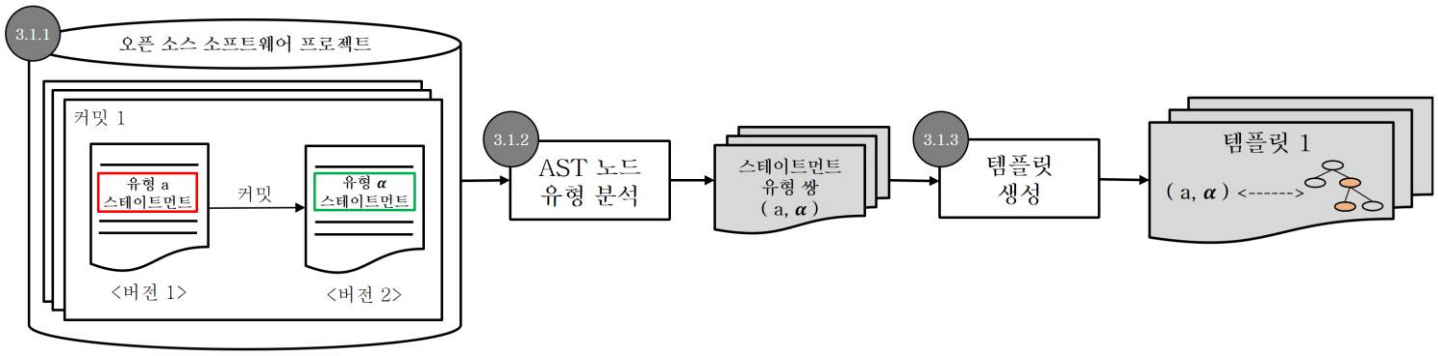


그림 1 SeeType의 템플릿 생성 프로세스

결과인 의심스러운 스테이트먼트의 AST 노드 유형과 동일한 유형을 가진 템플릿에 대해서만 패치 생성 및 변이(mutation)과정을 적용하므로 기존 APR 기법보다 옳은 패치 생성을 위한 탐사 효율이 높아질 수 있다. 또한 SeeType은 사람이 생성한 패치를 활용할 뿐만 아니라 해당 패치의 AST 유형 정보 또한 활용하기 때문에 기존보다 더욱더 많은 옳은 패치를 생성할 수 있다.

이 논문의 기여점은 다음과 같다.

- 기존의 사람이 작성한 패치에 대해 AST 노드 유형 정보를 활용한 새로운 탐사 영역 축소 기법 제안.
- 작은 단위 정보인 AST 기반 패치 생성으로 인한 높은 옳은 패치 생성률 기대.

이어지는 2장에서는 관련 연구를 설명하고 3장에서 본 논문에서 제안하는 SeeType 기법을 자세하게 설명한다. 4장에서는 실험 방법에 대해 논하고, 마지막으로 5장에서는 향후 연구와 함께 본 논문을 마무리 짓는다.

2. 관련 연구

2.1 과거 패치 이력 관련 연구

APR의 패치 생성 과정을 위해 과거에 작성 및 적용된 패치의 이력을 분석 및 활용하는 연구가 진행되었다. [10]은 사람이 작성한 패치를 수동으로 분석한 후 발견된 패턴으로 템플릿을 만들어 기존의 옳은 패치를 생성하지 못한 타깃 프로젝트(Target project)에 대해서 새로운 옳은 패치를 생성하였다. [8]은 오픈 소스 소프트웨어 저장소의 프로젝트에서 사람이 작성한 패치를 통해 옳은 패치의 특징을 추출하는 기계학습 모델을 생성했다. 생성된 기계학습 모델은 후보 패치를 우선 순위화하여 옳은 패치의 순위를 높이는 데 사용하여 이전 연구보다 더욱더 많은 옳은 패치를 생성하였다. 위의 연구를 통해 사람이 작성한 패치 이력에서 더욱더 많은 옳은 패치 생성이 가능함을 알 수 있다.

[11]은 과거 패치 이력을 코드 구조 관점에서 분석하였다. 과거 패치 간에 같은 프로젝트일 경우 약 90%의 AST 노드가 중복되고, 다른 프로젝트일 경우 약 40%가 겹치는 것을 확인했다. 따라서 타깃 프로젝트와 유사한 AST 및 코드 구조를 가진 프로젝트를 사용한다면 비슷한 과거 패치를 활용할 수 있기 때문에 효과적으로 옳은 패치를 생성 할 수 있다.

2.2 AST 노드 유형 정보 활용

기존부터 옳은 패치 생성을 위한 탐사 영역 구성 관련 연구가 진행되고 있다. [8]은 오픈 소스 소프트웨어 저장소의 사람이 작성한 패치를 통해 탐사 영역을 구성함으로써 옳은 패치를 생성했다. [9]은 프로그램 동작(Operation)과 패치 재료(patch ingredient)에 대한 경험적 규칙을 세워 규칙 기반 탐사 영역 축소 과정을 거친 후에 패치를 생성했다. 위의 연구들은 과거의 패치들을 기반으로 경험적인 규칙, 템플릿을 만들어 탐사 영역을 필터링하였다. 하지만 사람이 작성한 패치의 분석 정보 중 AST 노드 유형 정보를 활용하지는 않았다.

사람이 작성한 패치에 대한 분석을 내놓은 연구 [12]는 오픈 소스 소프트웨어 저장소에서 버그가 발생하는 AST 노드 유형 분석 시 스테이트먼트 관련 유형이 약 73%를 차지하고 있음을 밝혔다. 해당 73%를 전체로 두고 유형의 종류에 대해 분석 시 88%의 버그의 발생은 22가지의 스테이트먼트 유형 중 5가지에 의해서만 발생함을 밝혔다. 더불어 연구 [13]에서도 자바 기반의 패치를 분석하였는데 패치 간 중복되는 스테이트먼트 유형 및 패치 패턴이 존재한다고 밝혔다. 따라서 기존 패치에서 추출된 모든 템플릿을 탐사하지 않고 의심스러운 스테이트먼트의 AST 유형을 통해 필터링 된 템플릿으로 패치를 생성하는 것은 기존의 APR 기법보다 높은 효율을 얻을 수 있다.

3. SeeType : Search space reduction using Statement type

SeeType의 과정은 템플릿 생성(3.1절)과 해당 템플릿을 활용한 APR 과정(3.2절)으로 나뉜다. 템플릿 생성 과정에서는 과거의 패치 이력을 통해 템플릿을 생성하고, 템플릿을 활용한 APR 과정에서는 앞의 과정을 통해 생성된 템플릿을 활용하여 패치 생성 및 평가가 이루어진다.

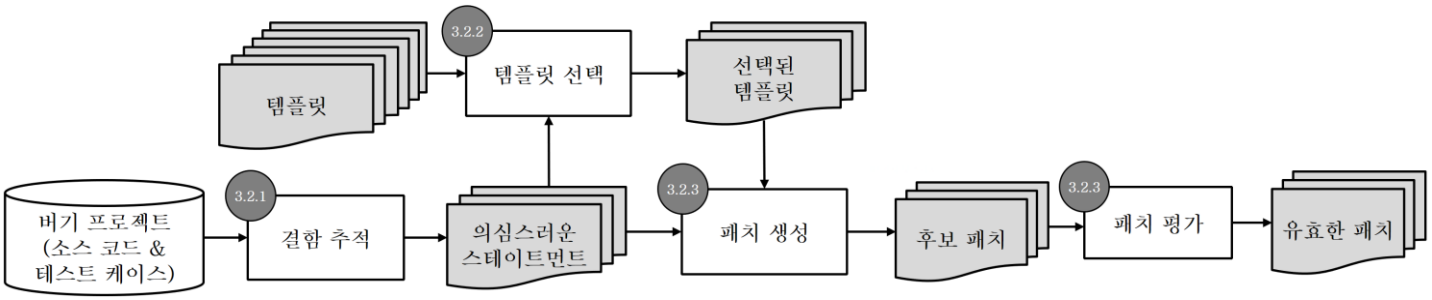
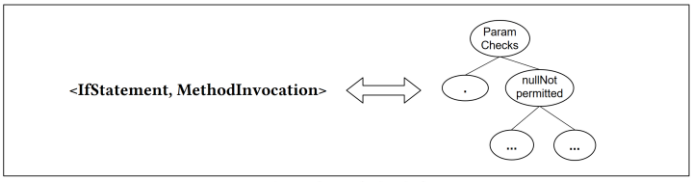


그림 2 SeeType의 템플릿 활용 APR 프로세스

```
commit af56b7dcd998392e95c49b0b3fd56e193c24efc0
src/main/java/org/jfree/chart/panel/CrosshairOverlay.java
@@ -97, 15 +99, 13 @@
-   if (crosshair == null) {
-       throw new IllegalArgumentException("Null 'crosshair' argument.");
-   }
+ ParamChecks.nullNotpermitted(crosshair, "crosshair");

AST type analyzing result :
<IfStatement, MethodInvocation>
```

a) 스테이트먼트 분석 결과



b) 템플릿 생성 예시

그림 3 템플릿 생성 과정 예시

3.1 템플릿 생성

그림 1은 SeeType의 템플릿 생성에 대한 프로세스를 보여주고 있다. 먼저 오픈 소스 소프트웨어 저장소의 프로젝트에서 사람이 작성한 패치를 수집한다(3.1.1절).

수집된 패치를 AST 형태로 변환 후, AST 노드 유형 분석을 한다(3.1.2절). 분석 결과로 패치 전, 후 스테이트먼트의 AST 노드 유형이 각각 나오게 되면 해당 쌍을 템플릿으로 취급하여 저장한다(3.1.3절).

3.1.1 과거 패치 이력 수집

과거 패치 이력 수집 단계에서는 템플릿을 생성하는데 필요한 과거의 패치들을 수집하게 된다. 우선 패치 생성 시 최대한 유사한 정보를 활용하기 위해 오픈 소스 소프트웨어 저장소 중 하나인 Github[14]에서 타깃 프로젝트와 유사한 프로젝트를 대상으로 수집한다. 이때 타깃 프로젝트와 동일한 프로그래밍 언어와 토픽(topic)을 가진 프로젝트를 유사한 프로젝트로 취급한다.

프로젝트를 선별한 이후, 패치를 수집하기 위해 타깃 프로젝트 및 선별된 프로젝트의 커밋(Commit)을 탐사한다. 커밋 메시지 중 패치 관련 키워드(예-fix, repair)를 포함하고 있는 경우 패치로 간주하여 선별한다.

3.1.2 스테이트먼트 유형 분석

스테이트먼트 유형 분석 단계에서는 템플릿 생성을 위해 패치를 AST로 변환 후 AST 노드 유형을 분석한다. AST 분석을 위해 Gumtree[15]라는 도구를 활용하는데, Gumtree는 코드를 AST로 변환하여 각각의 AST 노드에 대한 유형 분석 결과를 결과로 도출할 수 있다. AST 노드 유형에 대한 정의는 이클립스(Eclipse) 자바 문서에 미리 정의된 것을 사용한다[16] (정의의 일부를 부록에 첨부함).

기존 패치의 스테이트먼트 분석 시 패치 이전의 스테이트먼트, 패치 이후의 스테이트먼트를 각각 분석한다. 그리고 그림3의 a) 와 같이 **<패치 이전의 스테이트먼트 AST 노드 유형, 패치 이후의 스테이트먼트 AST 노드 유형>**의 형태로 쌍을 만든다. 이 쌍을 기반으로 템플릿을 도출하게 된다.

3.1.3 템플릿 생성

이 단계에서는 이전 단계(3.1.2절)의 결과를 가지고 템플릿을 생성하는 단계이다. 템플릿 활용 APR 과정에서 패치 생성 시 기존의 패치 정보 중 패치 이후의 정보를 가지고 패치를 생성한다. 따라서 패치 이후 코드의 AST를 템플릿과 연결 지어 최종적으로 템플릿을 생성한다. 예를 들어 그림 3의 b)처럼 스테이트먼트 유형 분석 단계의 결과가 <IfStatement, MethodInvocation>일 경우 **MethodInvocation의 코드 AST**와 연결 지어 하나의 템플릿으로 생각하게 된다.

자주 등장하는 패치의 유형을 패치 생성 시에 반영하기 위해 각각의 템플릿에 빈도수를 산출한다. 빈도수는 해당 템플릿이 수집한 패치의 목록에 등장한 횟수를 의미한다.

3.2 템플릿 활용 APR 과정

그림 2는 템플릿을 활용한 SeeType의 버그 수정 과정을 보여주고 있다. 결함 추적 과정(3.2.1절)에서는 타깃 프로젝트와 테스트 케이스를 이용해 의심되는 스테이트먼트를 도출한다. 이후 의심스러운 스테이트먼트의 AST 노드 유형 분석을 통해 템플릿을 선별한다(3.2.2절). 선별한 템플릿을 통해 패치를 생성하고 마지막으로 패치 평가를 거쳐 유효한 패치를 생성한다(3.2.3절).

3.2.1 결함 추적

결함 추적 단계에서는 타깃 프로젝트와 테스트 케이스를 통해 의심스러운 스테이트먼트를 도출한다. 결함 추적 알고리즘으로는 스펙트럼 기반 결함 추적 알고리즘 중 Tarantula[17]를 사용하되, 대상으로 하는 타깃 프로젝트에 따라 최적의 성능을 내기 위해 Ochiai[18], Jaccard[19]와 같은 다양한 알고리즘을 적용할 수 있다.

```
if (dataset != null) {

AST type analyzing result : IfStatement
```

a) 의심스러운 스테이트먼트 분석 결과

```
Buggy Statement Type : IfStatement
Template filtering result:
<IfStatement, MethodInvocation>
<IfStatement, VariableDeclarationFragment >
<IfStatement, ThisExpression>
...
```

b) 템플릿 필터링 예시

그림 4 템플릿 선택 과정 예시

표 1 Defects4j 실험 데이터 정보

프로젝트(식별자)	버그 수	라인 수 (단위: 천)	테스트 케이스
JfreeChart(Chart)	26	96	2,205
Closure compiler(Closure)	133	90	7,927
Apache commons-lang(Lang)	65	22	2,245
Apache commons-math(Math)	106	85	3,602
Mockito(Mockito)	38	11	1,457
Joda-Time(Time)	27	28	4,130
총합	**395**	**332**	**21,566**

3.2.2 템플릿 선택

템플릿 선택 단계에서는 패치 생성에 필요한 템플릿을 선별하여 탐사 영역을 줄이는 단계이다. 결함 추적 단계의 결과인 의심스러운 스테이트먼트에 대해 AST 노드 유형 분석을 진행하게 되는데 그림4의 a)와 같이 단일 결과로 나온다. 이후 템플릿 중 해당 AST 노드 유형을 통해 템플릿을 필터링해야 하는데, 결함 추적의 결과로 나온 의심스러운 스테이트먼트는 과거 패치와 비교할 때 패치 이전의 스테이트먼트와 상응하는 자리이다. 따라서 의심스러운 스테이트먼트의 AST 노드 유형과 패치 이전의 스테이트먼트의 AST 노드 유형이 같은 템플릿만 필터링한다. 예를 들어 그림4의 b)와 같이 의심스러운 스테이트먼트의 유형이 IfStatement라면 <IfStatement, MethodInvocation>, <IfStatement, VariableDeclarationFragment>와 같은 템플릿만 필터링 되게 된다.

3.2.3 패치 생성 및 평가

패치 생성 단계에서는 이전 단계에서 필터링 된 템플릿을 통해 패치를 생성하게 된다. 템플릿에 존재하는 빈도수 정보를 바탕으로 빈도수가 높은 템플릿부터 활용하여 패치를 생성한다. 구문적(Syntactic) 부분 확인을 위해 선택된 템플릿의 AST와 의심스러운 스테이트먼트의 AST에서 변수, 메소드 정보를 추출한다. 각각 추출된 정보를 통해 구문적 부분을 확인하고, 구문적 부분이 일치하면 템플릿을 활용하여 AST를 수정해 패치를 생성한다.

템플릿을 통해 후보 패치가 생성되지 않을 경우 변이 과정[20]을 거쳐 지정된 타임아웃(Timeout)까지 패치 생성을 시도한다.

생성된 후보 패치는 패치 평가과정을 통해 준비된 모든 테스트 케이스를 통과했을 때 유효한 패치로 판단한다. 그렇지 못할 경우 유효하지 못한 패치로 간주한다. 생성한 유효 패치의 개수가 지정한 수에 도달할 때까지 다음 순위의 템플릿을 사용하여 패치를 생성한다.

3.3 실행 과정

SeeType은 실험 시 조정할 수 있는 5개의 파라미터가 존재한다. 5개의 파라미터에는 결함 추적의 최대 결과 개수 N, 템플릿의 최대 개수 M, 유효 패치의 최대 개수 L, 타임아웃 관련 매개변수 T1, T2(단, T2 > T1)가 있다.

SeeType의 프로세스는 결함 추적 결과의 1순위부터 N 순위까지 순차적으로 진행된다. 결함 추적 결과의 1 순위부터 개별 추적 결과에 대해 템플릿을 최대 M개까지 추출한 후 패치를 생성한다. 템플릿들은 빈도수에 의해 우선순위가 부여되며, 1순위부터 패치를 생성한다. 패치를 생성하지 못했을 시 타임아웃 시간 T1 동안 변이과정을 수행하며, 타임아웃 이후에는 다음 순위의 템플릿으로 진행한다. M 순위까지의 템플릿 결과 활용 후 그다음 순위의 결함 추적 결과를 통해 위의 과정을 반복한다. 생성한 유효 패치가 L개가 되거나 전체 타임아웃 시간 T2에 도달할 경우 과정을 중단한다.

4. 실험계획

4.1 실험 데이터

4.1.1 벤치마크

평가에 사용되는 벤치마크는 자바 언어 기반 프로젝트들의 버그 및 테스트케이스를 모아놓은 Defects4j[21] 1.2.0버전을 사용한다.

표1의 항목 중 버그 수는 해당 프로젝트가 가지고 있는 총 버그 개수를 의미하고, 라인(line) 수는 프로그램의 크기를 천개 단위의 라인 개수로 나타낸 것이다. 마지막으로 테스트 케이스는 해당 프로젝트가 가지고 있는 총 테스트 케이스 수를 의미한다.

4.1.2 오픈 소스 프로젝트 수집

템플릿 생성을 위해 오픈 소스 소프트웨어 저장소에서 프로젝트를 수집하였다. 3.1.1절의 설명과 동일하게 타깃 프로젝트와 동일한 프로그래밍 언어와 토픽을 가진 프로젝트를 수집하였다. 그중 GitHub에서 유명도로 사용되는 척도 중 하나인 스타(star)가 많은 순으로 나열하여 Defects4j의 프로젝트당 5개의 프로젝트를 수집하였다. 해당 목록은 부록에 첨부하였다.

4.2 연구 질문

본 논문에서는 다음의 연구 질문에 대하여 답한다.

RQ1: SeeType은 기존의 다른 APR기법에 비해 얼마나 많은 옳은 패치를 생성할 수 있는가?

APR의 주요 목적은 옳은 패치를 생성하는 것이다. 따라서 첫 번째 연구 질문은 제안하는 기법의 옳은 패치 생성률을 살펴본다. 옳은 패치 생성률을 평가하기 위해서 SeeType이 생성한 유효한 패치가 옳은 것인지에 대한 평가가 필요하다. 이를 위해 생성한 유효 패치를 수동으로 확인하여 평가한다.

RQ2: SeeType은 옳은 패치 생성 시 얼마나 많은 시간이 소모되는가?

APR기법을 평가할 때 옳은 패치를 생성할 때 걸리는 시간 또한 중요하다. 따라서 두 번째로 제안 기법의 효율성을 살펴본다. 효율성은 두 가지 방법으로 확인한다. 먼저 각각의 기법들이 옳은 패치를 생성하는 데 걸리는 시간을 비교하고, 그다음 타임아웃의 길이에 따라 얼마나 많은 옳은 패치를 생성할 수 있는지를 비교한다.

RQ3: SeeType이 옳은 패치를 생성하지 못한 경우에는 어떤 것이 있는가?

제안하는 기법이 옳은 패치를 생성하지 못한 경우에는 크게 해당 버그에 대해 유효한 패치를 전혀 생성하지 못한 경우, 유효한 패치를 생성하였으나 수동으로 옳은 패치 평가 시 옳은 패치라고 판단되지 않는 경우가 있다. 따라서 향후 연구를 위해 어떤 버그에 대해서 옳은 패치를 생성하지 못했는지에 대해 분석한다.

연구 질문 1, 2의 비교 대상 기술은 기존의 패턴 기반의 APR 기법의 결과를 정리한 Tbar의 연구 결과[20]와 비교한다.

4.3 프로토타입 결과

SeeType 프로토타입에서 Defects4j의 Math 프로젝트를 대상으로 실험했을 때 기존 연구[7]보다 약 10%가량 새로운 버그에 대해 유효한 패치를 생성하는 것을 확인하였다.

5. 결론

탐사 기반 APR이 생성하는 패치의 질이 떨어진다는 점을 보완하기 위해 많은 연구가 진행되고 있다. 본 논문에서는 더욱더 높은 비율로 옳은 패치를 생성하기 위해 스테이트먼트 유형 기반 탐사 영역 축소 기법인 SeeType을 제안했다. 패치 생성 시 사람이 작성한 패치 정보와 스테이트먼트 유형에 따라 필터링 된 템플릿을 사용하기 때문에 더욱더 많은 옳은 패치와 높은 패치 생성 효율을 얻을 수 있을 것이다. 향후 연구로 기존의 연구에서 사용한 일반적인 벤치마크를 통해 실제 SeeType이 생성하는 패치에 대한 실험 결과를 실행 시간 및 생성하는 옳은 패치 수 측면에서 정량화하여 기존 연구와 비교하고자 한다.

Acknowledge

이 논문은 2019년도 정부(과학기술정보통신부)의 재원으로 한국연구재단-차세대정보컴퓨팅기술개발사업, 개인연구사업의 지원을 받아 수행된 연구임(No. 2017M3C4A7068179, No. 2019R1A2C2006411).

참고문헌

[1] Keigo Naitou, Akito Tanikado, Shinsuke Matsumoto, Yoshiki Higo, Shinji Kusumoto, Hiroyuki Kirinuki, Toshiyuki Kurabayashi, and Haruto Tanno. Toward introducing automated program repair techniques to industrial softwaredevelopment. In Proceedings of the 26th Conference on Program Comprehension, ICPC '18, pages 332–335, New York, NY, USA, 2018. ACM.

[2] L. Gazzola, D. Micucci, and L. Mariani. Automatic software repair: A survey. IEEE Transactions on Software Engineering, 45(1):34–67, Jan 2019.

[3] Edward K. Smith, Earl T. Barr, Claire Le Goues, and Yuriy Brun. Is the cure worse than the disease? overfitting in automated program repair. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, pages 532–543, New York, NY, USA, 2015. ACM.

[4] Ming Wen, Junjie Chen, Rong Wu, Dan Hao, Shing-Chi Cheung. An empirical analysis of the influence of fault space on search-based automated program repair. arXiv preprint arXiv: 1707.05172, 2017.

[5] Zichao Qi, Fan Long, Sara Achour, and Martin Rinard. An analysis of patch plausibility and correctness for generate and validate patch generation systems. In Proceedings of the 2015 International Symposium on Software Testing and Analysis, ISSTA 2015, pages 24–36, New York, NY, USA, 2015. ACM.

[6] F. Long and M. Rinard. An analysis of the search spaces for generate and validate patch generation systems. In 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), pages 702–713, May 2016.

[7] Jiajun Jiang, 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, ISSTA 2018, pages 298–309, New York, NY, USA, 2018. ACM.

[8] Fan Long and Martin Rinard. Automatic patch generation by learning correct code. SIGPLAN Not., 51(1):298–312, January 2016.

[9] Yuan Yuan and Wolfgang Banzhaf. ARJA: automated

repair of java programs via multi-objective genetic programming. CoRR, abs/1712.07804, 2017.

[10]Dongsun Kim, Jaechang Nam, Jaewoo Song, and Sunghun Kim. Automatic patch generation learned from human-written patches. In Proceedings of the 2013 International Conference on Software Engineering, ICSE '13, pages 802-811, Piscataway, NJ, USA, 2013. IEEE Press.

[11]Hao Zhong and Na Meng. Towards reusing hints from past fixes. Empirical Software Engineering, 23(5):2521-2549, Oct 2018.

[12]K. Liu, D. Kim, A. Koyuncu, L. Li, T. F. Bissyandé, and Y. Le Traon. A closerlook at real-world patches. In 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), pages 275-286, Sep. 2018.

[13]Eduardo C. Campos and Marcelo de A. Maia. Discovering common bug-fixpatterns: A large-scale observational study. Journal of Software: Evolution andProcess, 31(7): e2173, 2019. e2173 smr, 2173.

[14]Github, 2020 년 1 월 2 일 접속, https://www.github.com

[15]Jean-Rémy Falleri, Floréal Morandat, Xavier Blanc, Matias Martinez, and Martin Monperrus. Fine-grained and accurate source code differencing. In Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering, ASE '14, pages 313-324, New York, NY, USA, 2014. ACM.

[16]ASTNode, clipse, 2020년 1월 2일 접속, https://help.eclipse.org/2019-12

[17]J.A. Jones and M. J. Harrold. Empirical evaluation of the Tarantula automatic fault-localization technique. In Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering. ACM, pages 273-282, 2005.

[18]R. Abreu, P. Zoeteweij, and A. J. c. Van Gemund. An evaluation of similarity coefficients for software fault localization. In 2006 12th Pacific Rim International Symposium on Dependable Computing, pages 39-46, Dec 2006.

[19]M.Y. Chen, E. Kiciman, E. Fratkin, A. Fox, E. Brewer. Pinpoint: Problem determination in large, dynamic internet services. In Proceedings International Conference on Dependable Systems and Networks. IEEE, pages 595-604, 2002.

[20]Kui Liu, Anil Koyuncu, Dongsun Kim, Tegawende F. Bissyande. TBar: Revisiting Template-based Automated Program Repair. In Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2019, 2019.

[21]René Just, Darioush Jalali, and Michael D. Ernst. Defects4j: A database of existingfaults to enable controlled testing studies for java programs. In Proceedings of the 2014 International Symposium on Software Testing and Analysis, ISSTA 2014, pages 437-440, New York, NY, USA, 2014.

부록

부록 A. ASTNode 필드 정보(일부)

수정자와 자료형	필드(field)	설명
static int	ANNOTATION_TYPE_DECLARATION	AnnotationTypeDeclaration 형의 노드를 나타내는 노드 형 정수.
static int	ARRAY_ACCESS	ArrayAccess 형의 노드를 나타내는 노드 형 정수.
static int	ARRAY_TYPE	ArrayType 형의 노드를 나타내는 노드 형 정수.
static int	ASSIGNMENT	Assignment 형의 노드를 나타내는 노드 형 정수.
static int	BOOLEAN_LITERAL	BooleanLiteral 형의 노드를 나타내는 노드 형 정수.
static int	BREAK_STATEMENT	BreakStatement 형의 노드를 나타내는 노드 형 정수.
static int	CAST_EXPRESSION	CastExpression 형의 노드를 나타내는 노드 형 정수.
static int	CLASS_INSTANCE_CREATION	ClassInstanceCreation 형의 노드를 나타내는 노드 형 정수.
static int	CONTINUE_STATEMENT	ContinueStatement 형의 노드를 나타내는 노드 형 정수.
static int	FOR_STATEMENT	ForStatement 형의 노드를 나타내는 노드 형 정수.
static int	IF_STATEMENT	IfStatement 형의 노드를 나타내는 노드 형 정수.
static int	IMPORT_DECLARATION	ImportDeclaration 형의 노드를 나타내는 노드 형 정수.
static int	INFIX_EXPRESSION	InfixExpression 형의 노드를 나타내는 노드 형 정수.
static int	INITIALIZER	Initializer 형의 노드를 나타내는 노드 형 정수.
static int	METHOD_DECLARATION	MethodDeclaration 형의 노드를 나타내는 노드 형 정수.
static int	METHOD_INVOCATION	MethodInvocation 형의 노드를 나타내는 노드 형 정수.
static int	METHOD_REF	MethodRef 형의 노드를 나타내는 노드 형 정수.
static int	PARAMETERIZED_TYPE	ParameterizedType 형의 노드를 나타내는 노드 형 정수.
static int	PARENTHESIZED_EXPRESSION	ParenthesizedExpression 형의 노드를 나타내는 노드 형 정수.
static int	POSTFIX_EXPRESSION	PostfixExpression 형의 노드를 나타내는 노드 형 정수.
static int	PREFIX_EXPRESSION	PrefixExpression 형의 노드를 나타내는 노드 형 정수.
static int	TYPE_DECLARATION	TypeDeclaration 형의 노드를 나타내는 노드 형 정수.
static int	TYPE_DECLARATION_STATEMENT	TypeDeclarationStatement 형의 노드를 나타내는 노드 형 정수.
static int	TYPE_METHOD_REFERENCE	TypeMethodReference 형의 노드를 나타내는 노드 형 정수.
static int	TYPE_PARAMETER	TypeParameter 형의 노드를 나타내는 노드 형 정수.
static int	VARIABLE_DECLARATION_FRAGMENT	VariableDeclarationFragment 형의 노드를 나타내는 노드 형 정수.
static int	WHILE_STATEMENT	WhileStatement 형의 노드를 나타내는 노드 형 정수.

부록 B. 템플릿 생성을 위한 오픈 소스 프로젝트 목록

- JfreeChart(Chart)

프로젝트명	설명
OpenRefine/OpenRefine	자바 기반 데이터 관리 도구
jtablesaw/tablesaw	자바 데이터 프레임 및 시각화 라이브러리
shzlw/poli	SQL 보고 응용 프로그램
AnyChart/AnyChart-Android	안드로이드 어플리케이션 기반 데이터 시각화 라이브러리
KnowageLabs/Knowage-Server	빅 데이터 시스템 관리 라이브러리

- Closure compiler(Closure)

프로젝트명	설명
CalebFenton/simplify	안드로이드 어플리케이션 코드 최적화 라이브러리
Kyson/AndroidGodEye	안드로이드 어플리케이션 성능 모니터링 라이브러리
Kiegroup/optaplanner	인공지능 기반 constraint solver 엔진
Sable/soot	자바 최적화 프레임워크
amitshekhariitbhu/GlideBitmapPool	비트맵 메모리 재사용을 위한 메모리 관리 라이브러리

- Apache commons-lang(Lang), Apache commons-math(Math)

프로젝트명	설명
apache/commons-io	I/O 관련 기능 지원 라이브러리
apache/commons-collections	자바 콜렉션 처리 관련 라이브러리
apache/commons-pool	객체 풀링 API 및 객체 풀 구현 제공 라이브러리
Siznax/wptools	위키피디아(Wikipedia) 데이터 추출 프로그램
apache/commons-codec	자바 인코더, 디코더 구현 라이브러리

- Mockito(Mockito)

프로젝트명	설명
powermock/powermock	테스트 불가 코드 단위화 프레임워크
ihsanbal/LoggingInterceptor	요청, 응답을 위한 logger 가 있는 Okhttp 인터셉터 라이브러리
fabioCollini/DaggerMock	Dagger 2 오브젝트 오버라이드(override) 지원 라이브러리
yale8848/RetrofitCache	캐시 추가 지원 라이브러리
OpenConext/Mujina	OpenSAML 및 자바 Spring Boot 기반 서비스 제공 라이브러리

- Joda-Time(Time)

프로젝트명	설명
JodaOrg/joda-money	화폐 저장 관련 클래스 라이브러리
JodaOrg/joda-beans	JavaBeans 관련 프레임 워크
Gkopff/gson-jodatime-serialisers	Joda Time 엔티티(Entity) 처리 라이브러리
JodaOrg/joda-convert	표준 문자열 형식과의 변환을 지원하는 자바 라이브러리
JodaOrg/joda-time-hibernate	JDK 날짜 및 시간 관련 라이브러리