기존 자바 파서 확장 기반의 코드 정적 분석기 구현

박지훈*, 박보경, 이근상, 김영철
홍익대학교 소프트웨어공학연구실
e-mail: {pjh, park, yi, bob}@selab.hongik.ac.kr

Implementing A Code Static Analysis based on the Java Parser

Jihoon Park*, Bokyung Park, Keunsang Yi, R. Youngchul Kim
SE Lab, Hongik University

요 약
현재 많은 테스팅 기법으로도 생산되는 결과물들의 검토에 오류 발생을 예측하기 힘들다. 기존 오픈 소스 정적 분석 도구들(Source Navigator)은 분명한 정보를 제공하여 원하는 내부 정보를 추출하기 어렵다. 이를 해결하기 위해, 기존 오픈 소스의 자바 파서의 개선을 통해, 코드 내부 품질 측정을 고려하고자 한다. 즉 기존 자바 파서 개선 기반의 ‘추상구문트리’로 변환된 코드로 ‘직접 코드 정보 추출’ 방안의 구현이다. 이를 통해, 기존의 SNDB보다 더 많은 코드 정보 추출으로 코드 내부 품질 측정이 더 수월할 것을 기대한다.

1. 서론
모든 분야의 공학에서는 각 분야별로 테스트 기법을 가지고 있다. 제대로 테스트 되지 않은 결과물들은 언제 어떤 문제가 발생할지 예측하기 힘들어진다. 2013년 12월, 네덜란드의 알스테르담 시는 1억 7천만 유로의 손해를 입었다. 문제는 180만 유로의 주택 보조금 지급하는 소프트웨어의 단위가 100배나 크게 설정된 사실을 파악 못했기 때문이다 [4]. 소프트웨어 사용 전에 테스트만 잘 수행해도 이런 큰 손해를 막을 수 있었다.


본 논문의 순서는 다음과 같다. 1장에서는 서론, 2장은 AST(추상구문트리)에 대한 설명이다. 3장은 소스 내비게이터 DB와 자바 파서 DB의 차이점이다. 4장은 결론 및 향후 연구이다.

2. 추상화 구문 트리 (Abstract Syntax Tree)
추상 구문 트리는 프로그래밍 언어로 작성된 소스 코드의 추상 구문 구조를 표현한 것이다.

(그림 1) 추상구문트리로 변환된 Java 코드
3. 기존연구와의 파생 도구 비교

기존연구에서는 소스 코드를 가시화하는 자동화 도구에서 파생분할을 בן포소스인 소스 내비게이터로 사용하였다. 그림 2는 소스 내비게이터가 자동으로 추출해주는 소스코드의 데이터베이스와 자바 파서로 소스코드의 정보를 직접 추출한 데이터베이스를 비교한 그림이다. 파란색은 기존의 값 을 한 단계 더 해석한 것이고, 노란색은 새로 추가된 값.

(그림 2) 소스 내비게이터와 자바 파서의 데이터베이스 비교

<table>
<thead>
<tr>
<th>Source Navigator의 정보</th>
<th>Java Parser의 정보</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>Object</strong></td>
</tr>
<tr>
<td>C:/Java/Visualization/t...</td>
<td>Table</td>
</tr>
<tr>
<td>SNDB_CL</td>
<td>Field</td>
</tr>
<tr>
<td>NAME</td>
<td>Field</td>
</tr>
<tr>
<td>START_LINE_NO</td>
<td>Field</td>
</tr>
<tr>
<td>PATH</td>
<td>Field</td>
</tr>
<tr>
<td>END_LINE_NO</td>
<td>Field</td>
</tr>
<tr>
<td>ATTRIBUTES</td>
<td></td>
</tr>
<tr>
<td>SNDB_IV</td>
<td>Table</td>
</tr>
<tr>
<td>CLASS_NAME</td>
<td>Field</td>
</tr>
<tr>
<td>VARIABLE_NAME</td>
<td>Field</td>
</tr>
<tr>
<td>START_LINE_NO</td>
<td>Field</td>
</tr>
<tr>
<td>PATH</td>
<td>Field</td>
</tr>
<tr>
<td>END_LINE_NO</td>
<td>Field</td>
</tr>
<tr>
<td>ATTRIBUTES</td>
<td></td>
</tr>
<tr>
<td>SNDB_MD</td>
<td>Table</td>
</tr>
<tr>
<td>CLASS_NAME</td>
<td>Field</td>
</tr>
<tr>
<td>VARIABLE_NAME</td>
<td>Field</td>
</tr>
<tr>
<td>START_LINE_NO</td>
<td>Field</td>
</tr>
<tr>
<td>PATH</td>
<td>Field</td>
</tr>
<tr>
<td>END_LINE_NO</td>
<td>Field</td>
</tr>
<tr>
<td>ATTRIBUTES</td>
<td></td>
</tr>
<tr>
<td>SNDB_BY</td>
<td>Table</td>
</tr>
<tr>
<td>REFERRED_CLASS</td>
<td>Field</td>
</tr>
<tr>
<td>REFERRED_SYM</td>
<td>Field</td>
</tr>
<tr>
<td>REFERRED_TYPE</td>
<td>Field</td>
</tr>
<tr>
<td>REFER_CLASS</td>
<td>Field</td>
</tr>
<tr>
<td>REFER_SYMBOL</td>
<td>Field</td>
</tr>
<tr>
<td>REFER_TYPE</td>
<td>Field</td>
</tr>
<tr>
<td>CALLER</td>
<td>Field</td>
</tr>
<tr>
<td>CALLED</td>
<td>Field</td>
</tr>
<tr>
<td>SNDB_IN</td>
<td>Table</td>
</tr>
<tr>
<td>SUB_CLASS_NAME</td>
<td>Field</td>
</tr>
<tr>
<td>SUPER_CLASS_NAME</td>
<td>Field</td>
</tr>
<tr>
<td>START_LINE_NO</td>
<td>Field</td>
</tr>
<tr>
<td>END_LINE_NO</td>
<td>Field</td>
</tr>
<tr>
<td>ATTRIBUTES</td>
<td></td>
</tr>
</tbody>
</table>
터에서 자동으로 생성된 SNDB의 소스코드 데이터는 그림 2의 원쪽 데이터이다. 데이터를 보면 소스 내비게이터에서 파생하여 추출해야 한 코드의 정보는 총 6개의 테이블에 저장되어 있다. 그리고 새로 구성한 자바 파서를 이용하여 직접 추출한 소스코드 데이터는 그림 2의 오른쪽 데이터와 같다. 데이터를 보면 자바 파서로 파싱하여 얻은 AST 데이터들은 5개의 테이블에 정해진 것을 볼 수 있다.

먼저 소스 내비게이터의 데이터 중 SNDB_CL 테이블에는 클래스의 이름, 시작 라인 넘버, 파일의 경로, 마지막 라인 넘버, 접근자의 정보가 들어있다. SNDB_IV 테이블에는 인스턴스 변수가 속해있는 클래스, 인스턴스 변수의 이름, 시작 라인 넘버, 파일의 경로, 마지막 라인 넘버, 접근자의 정보가 들어있다. SNDB_LV 테이블에는 로컬변수의 이름, 시작 라인 넘버, 파일의 경로, 마지막 라인 넘버, 접근자의 정보가 들어있다. SNDB_MD 테이블에는 메서드가 속해있는 클래스, 메서드의 이름, 시작 라인 넘버, 파일 경로, 마지막 라인 넘버, 접근자, 리턴 타입, 파라미터의 타입, 마라미터의 이름이 들어있다. SNDB_BY 테이블에는 호출당하는 메서드나 변수의 이름과 클래스, 타입, 호출하는 메서드의 이름과 클래스, 타입, 메서드 정보, 라인 넘버, 파일 경로, 호출당하는 메서드의 파라미터의 타입이 들어있다. SNDB_IN 테이블에는 서브클래스의 이름, 슈퍼클래스의 이름, 시작 라인 넘버, 파일 경로, 마지막 라인 넘버, 접근자의 정보가 들어있다.

그에 비해 자바 파서로 추출한 테이블 중 JPDB_CLASS 테이블에는 SNDB의 0x404자리의 ASCII 코드가 아닌 스트링으로 public, protected, private, static, final을 나타내었다. 이는 아스키코드를 스타일 값으로 변환한 후 한 번 더 스크립트하여 접근자를 정해내서 했던 단계를 줄여준다. 그리고 SNDB_IN 테이블에 들어져 있던 상속정보를 extend와 implement를 넣어 저장함으로써 캐테그리가 더 간단해졌다. JPDB_METHOD 테이블에도 마찬가지로 접근자의 정보를 스타일 문자열로 가시화하였고 SNDB_MD 테이블에서는 메서드의 리턴 타입은 있었지만 메서드의 리턴 값은 나와 있지 않았다. 하지만 JPDB_METHOD 테이블에서는 메서드의 리턴 타입과 리턴 값까지 저장되어, 리턴 값에 있는 변수이름을 이용하여 어떠한 값이 리턴되는지지도 확인할 수 있다. 추가적으로 메서드 안의 while, if, switch 문의 개수를 파악하여 이용할 수도 있도록 하였다. JPDB_INSTANCE_VAR 테이블도 마찬가지로 접근자의 정보를 스타일 문자열로 가시화하였다. 그리고 SNDB_IV 테이블에서는 인스턴스 변수의 초기화 값을 알 수 없었지만 JPDB_INSTANCE_VAR 테이블에서는 각 인스턴스 변수들의 초기화 값을 알 수 있다. JPDB_LOCAL_VAR 테이블은 로컬변수의 정보가 들어있다. 로컬변수는 접근자를 가지고 있지 않기 때문에 각 소스SNDB_LV의 Attribute값을 삭제하
고 리턴 타입과 초기화 값을 추가하였다. JPDB REFERBY 테이블에서는 각 멤버간의 출력정보를 나타낸다. 출력하는 클래스와 변수 또는 메서드의 이름, 출력받는 클래스와 변수 또는 메서드의 이름, 출력하는 멤버의 타입, 라인 넘버, 파일 경로가 들어있다. 기존의 SDBY 테이블의 Access값과 Argument값은 사용 안 되어 삭제하였다.

그럼 3은 실제 Tool-Chain 코드 소스 내비게이터의 자바 파서를 이용하여 데이터를 추출한 것이다. 그림 3의 ①,②번은 SDBY, ③번은 JPDB이다. 두 가지 메시드의 로컬 변수에 대한 테이블이다. ①번에서는 메서드의 이름이 나와있지 않고 접근자가 아스키코드로 표현되었다. 하지만 ②번에서는 로컬변수가 속해있는 메서드의 이름이 리턴 타입, 초기값이 표현되어있다. ①번은 로컬변수가 객체를 몇 번이나 생성하는지 알 수 없지만 ②번은 초기값 값이 이용하여 객체가 몇 번 생성되는지 찾아낼 수 있다. ③번에서 는 메서드의 시작라인 넘버만 나와있지만 ②번에서는 메서드의 전체길이도 확인할 수 있다. ②번은 메서드가 void가 아닌때 어떤 변수를 리턴하는지 확인할 수 있지만 ④번은 RETURN_VALUE값으로 다른 객체에 얼마나 많은 위임을 시도하는지 확인할 수 있다.

4. 결론 및 향후계획

정적 분석은 대상 소프트웨어를 실제적으로 실행하지 않는 상태에서 수행하는 것이다. 그래서 실질적인 오류를 발견하기에 적합하지 않다. 하지만 보통 리뷰와 마찬가지로 장애(Failures)보다는 결함(Defects)을 발견하는데 중점을 두는 동적 테스팅에서 발견하기 어려운 문제점을 찾아낼 수 있다. 또한 코드 개발 중에 수정이 가능하다. 실형에 유지보수성을 향상시킬 수 있음으로써 소프트웨어의 품질 향상에 기여한다.

기존연구에서 사용하던 오픈소스인 정적분석도구 소스 내비게이터로 데이터 추출에 한계를 느꼈다. 그래서 본 논문에서는 소스 내비게이터 부분을 또 다른 오픈 소스인 자바 파서로 교체하였다. 하지만 자바 파서는 소스 내비게이터와 같은 정적분석한 결과만 자동으로 데이터베이스에 저장하지 않는다. 그래서 자바 파서 내의 자바 코딩 파싱 기능을 이용하여 직접 데이터베이스를 자동 생성할 수 있는 정적 분석 도구를 구현하였다. 즉 자바 파서는 소스코드의 모든 부분을 추상구문도로 변환할 수 있다. 소스 내비게이터가 찾아내지 못하는 소스코드의 더 깊은 정보를 추출할 수 있다. 그러므로 기존연구의 소스 내비게이터 기반 Tool-Chain(1,2,3)보다 더 다양한 소스 코드의 복잡도를 찾아낼 수 있다.

참고문헌

춘계학술발표대회
논문집

일자 2017년 4월 28일(금) ~ 29일(토)
장소 제주대학교 아라캠퍼스
주최 한국정보처리학회
주관 제주대학교
공조 스마트그리드와 청정에너지 융복합산업 인력양성사업단
후원 한국과학기술단체총연합회
협찬 NAVER, KCC정보통신

학술대회 홈페이지
http://www.kips.or.kr/2017spring
180. PCR Gel 전기영동 이미지 분석을 위한 레인검출 및 추적 알고리즘 KIPS_C2017A0060

181. 분야별로 분류된 콘텐츠를 제공하는 어플리케이션의 설계와 구현 KIPS_C2017A0065

182. BASMATI 플랫폼을 위한 애플리케이션 제어기 설계 KIPS_C2017A0080

183. 전원 무결성을 이용한 전자식 엔진 마운트 시스템 개발 KIPS_C2017A0095

184. 차량 전장용 운영체제의 주기적 테스트 검증을 지원하는 테스트 케이스 생성기 KIPS_C2017A0135

185. 공공부문 IT프로젝트 발주시스템 선정화에 관한 연구 - SW분할발주 제도를 중심으로- KIPS_C2017A0144

186. 테스트 데이터 자동 생성을 위한 입력 변수 스토리지와 진화 알고리즘 적용 방법 KIPS_C2017A0145

187. 구직자용 후천 알고리즘 연구 KIPS_C2017A0149

188. 의료기기 소프트웨어 위험관리를 위한 PEMS 개발생명주기 기반 위험관리 항목 관리 분석 KIPS_C2017A0153

189. 대사중추후 관을 위한 생활습관정보 관리 시스템의 개발 KIPS_C2017A0155

190. 모바일 1인칭 연기 시뮬레이션 앱 설계 및 구현 KIPS_C2017A0161

191. 센서 활용 어플리케이션을 위한 동적 테스트 도구의 요구사항 KIPS_C2017A0163

192. Test Case Generation Technique for IoT Mobile Application KIPS_C2017A0182

193. 정적/동적 분석 기반의 재사용 메트릭과 가시화 구축 KIPS_C2017A0196

194. 백색광 주사간섭을 활용한 3차원 표면 형상 분석 시스템 개발 KIPS_C2017A0197

195. 코드 가시화의 서비스 모듈화 KIPS_C2017A0200

196. CDLine: A Cloud Service for Dynamic Analysis of JavaScript Code KIPS_C2017A0210

197. 이층 Cloud Service Brokerage에 적응 가능한 실시간 모니터링 시스템 연구 KIPS_C2017A0219

198. 기존 자바 파서 확장 기반의 코드 정적 분석기 구현 KIPS_C2017A0224