논문 2018-2-4

공통 토큰에 기반한 서로 다른 언어의 유사성 검사

홍성문*, 김현하**, 이제형***, 박성우***, 모지환***, 도경구***†

Cross-Language Clone Detection based on Common Token

Sung-Moon Hong*, Hyunha Kim**, Jaehyung Lee***,
Sungwoo Park***, Ji-Hwan Mo***, Kyung-Goo Doh***†

요 약

서로 다른 언어로 작성된 소스코드의 유사성 검사는 주로 요약구문트리를 기반으로 비교를 수행한다. 하지만 대규모의 소스코드를 실용적인 수준으로 비교하려면 토큰수준 기반에서 작동하는 유사성 검사 기술이 필요하다.

본 연구에서는 서로 다른 언어에서 생성되었지만 같은 의미를 지닌 토큰을 표현할 수 있는 공통 토큰을 정의하고, 소스코드에서 언어별 처리 과정을 거쳐 생성한 공통 토큰의 나열을 입력으로 소스코드의 유사성 검사를 수행하는 방법을 제안한다. 한국저작권위원회의 표절검사 도구 exEyes를 사용해서 서로 다른 언어로 작성된 동일한 코드를 대상으로 실험한 결과, 제안한 방법을 사용했을 때, 유사성 평가 성능이 향상됨을 보였다.

Abstract

Tools for detecting cross-language clones usually compare abstract-syntax-tree representations of source code, which lacks scalability. In order to compare large source code to a practical level, we need a similarity checking technique that works on a token level basis.

In this paper, we define common tokens that represent all tokens commonly used in programming languages of different paradigms. Each source code of different language is then transformed into the list of common tokens that are compared. Experimental results using exEyes show that our proposed method using common tokens is effective in detecting cross-language clones.

한글키워드 : 언어간 클론 탐지, 클론 검사, 코드 클론, 소스코드 표절, 공통 토큰

keywords : cross-language clone detection, clone checking, code clone, common tokens

1. 서 론

소스코드 표절은 원본 자료의 출처를 분명히 밝히지 않고 자신의 것처럼 사용하는 행위를 말한다. 학생이 제출하는 과제부터 산업 현장의 프로그램 일부나 전체에 이르기까지 소스코드 표절 범위는 매우 다양하다. 소스코드 표절 문제는 인터넷으로 소스코드의 전달과 공유가 쉬워지면서 더 만연해지고 있으며, 공개된 오픈 소스코드의

* 한양대학교 ERICA 대학원 컴퓨터공학과
** 소프토피아(주)
*** 한양대학교 ERICA 소프트웨어학부
† 교신저자: 도경구(email: doh@hanyang.ac.kr)
접수일자: 2018.11.30. 심사완료: 2018.12.10.
게재확정: 2018.12.21.

www.kci.go.kr

공개를 통해 사용, 복제, 수정, 배포가 자유롭다는 강점을 바탕으로 전 세계적으로 그 영역이 확대되고 있다.

소스코드의 표절을 탐지하는 소스코드의 유사성 검사는 문자열, 토큰, 트리, 의존성그래프 등 여러 가지 방법이 존재한다. 문자열과 토큰 기반은 빠른 검사가 가능하지만 그만큼 분석의 정확도가 낮다. 트리 기반이나 의존성 그래프 기반 분석은 더 정확한 탐지가 가능하지만 그만큼 분석시간이 오래 걸린다. 이런 점에 실용적인 도구는 문자열 또는 토큰을 주로 사용한다.

각 분석기법들은 기본적으로 같은 언어로 작성된 소스코드를 대상으로 분석한다는 전제를 가지고 있다. 하지만 소스코드의 표절은 같은 언어가 아니라 다른 언어를 대상으로도 이루어질 수 있다. 다른 언어로 작성된 코드를 그대로 대상 언어로 옮겨 쓰는 작업은 과제를 베끼는 수준이 그치지 않고 이직 후 기존 직장에서 작성했던 코드를 옮긴 직장에서 사용하는 언어로 그대로 옮겨 사용해서 법적인 분쟁을 다투는 경우까지 다양하다.

서로 다른 언어로 작성된 코드는 그 의미가 동일하더라도 문법의 모양이 다르기 때문에 기존의 방법으로는 정확한 분석이 불가능하다. 이러한 약점은 토큰기반의 분석이 가지고 있는 한계이기 때문에, 대부분 요약문법트리 기반으로 수행하였다[1-3]. 하지만 요약문법트리 기반의 유사성 검사는 토큰기반의 분석방법에 비해 수행 속도가 매우 느리기 때문에 대단위의 소스코드 표절을 판정하는 경우에는 한국저작권위원회의 exEyes[4]와 같은 토큰 기반의 도구가 사용된다.

본 논문에서는 서로 다른 언어로 작성된 소스코드의 유사성을 검사하기 위해 기존 토큰기반의 분석 방법을 사용하되, 토큰의 나열을 공통 토큰의 나열로 변환한 후 비교하는 방법을 제안한다.

2. 서로 다른 언어 비교의 문제점

그림 1은 Java와 Python으로 각각 작성한 의미가 동일한 프로그램 조각이다. 맨눈으로 검사해보면 동일한 프로그램임을 바로 확인할 수 있다. 그러나 이 두 소스코드를 exEyes(Ver. 4.1)와 같은 도구로 비교하면 동일한 부분이 없는 것으로 나타난다.

라인	소스코드
38	while ((line = br.readLine()) != null) {
39	final int idx = line.indexOf('=');
40	if (idx >= 0) {
41	final String key = line.substring(0, idx);
42	final String value = line.substring(idx + 1);
43	envVars.setProperty(key, value);
44	}
45	}
46	return envVars;
47	}
48	}

라인	소스코드
30	while (line = br.readLine()) != None:
31	idx = line.indexOf('=')
32	if idx >= 0:
33	key = line.substring(0, idx)
34	value = line.substring(idx + 1)
35	envVars.setProperty(key, value)
36	return envVars

그림 1. 의미가 동일한 Java와 Python 프로그램
Fig 1. Semantically Identical Java and Python Code Snippets

맨눈으로는 프로그램의 실행 흐름을 관찰하거나 모양새를 살펴보면서 유사한 코드라고 쉽게 판단 할 수 있지만 탐지 도구들은 정해놓은 조건이나 룰을 엄격하게 적용하면서 유연성이 떨어지기 때문에 조금이라도 맞지 않으면 다르다고 판단할 수 있기 때문이다. 위 소스코드 조각에서는 여닫는 괄호의 수, Java에서 문장을 구분하는데

www.kci.go.kr

사용하는 쌍반점(;), 타입 및 한정자 등의 모양이나 빈도가 다르기 때문에 소스코드의 유사성을 탐지하는 기법에 따라 다른 결과가 나오게 된다.

가장 기본적인 문자열 기반의 분석[5,6]은 문자열을 문자 단위로 비교하기 때문에 서로 다른 언어로 작성된 프로그램의 유사성 검사에는 부적절하다. 토큰기반의 분석[7,8]은 토큰 단위로 쪼갠 후에 토큰열을 비교하는데, 토큰 단위로 비교하더라도 그림 1의 예제에서 볼 수 있듯이 Python은 Java와 달리 블록의 구조를 들여쓰기로 구분하기 때문에 여닫는 괄호가 쓰이지 않고, 반대로 타입이나 final과 같은 한정자가 사용되지 않기 때문에 유사성 판정에서 빗나가게 된다. 트리 기반[9,10]이나 의존 그래프 기반[11,12]과 같은 분석 방법은 앞선 두 방법에 비해 프로그램의 구문 구조에 좀 더 집중할 수 있어서 모양에 집중할 수 있지만, 분석이 상대적으로 오래 걸릴 뿐 아니라 다양한 언어 도입을 위해 각 언어에 맞는 구문분석이 반드시 필요하기 때문에 비용 부담이 있다. 이에 본 논문에서는 분석속도에 이점이 있는 토큰기반 방식으로 서로 다른 언어로 작성된 소스코드의 유사성을 검사할 수 있도록 새로운 방식을 제안한다.

3. 공통 토큰

전문가라면 처음 접하는 프로그래밍 언어로 작성된 프로그램이라 할지라도 경험과 지식만으로 어느 정도 프로그램의 의미를 파악할 수 있다. 이러한 아이디어에서 착안하여 대부분의 프로그래밍 언어에서 공통적으로 사용하는 토큰을 망라하여 공통토큰들을 정한다.

그림 2는 서로 다른 언어로 작성된 언어에서 공통 토큰의 나열을 얻어내는 과정이다. 먼저 언어에 의존적이지 않은 넓은 의미의 공통 어휘분석도구 (Common Lexical Analyzer)를 사용해서 토큰의 나열을 생성한다. 언어에 의존적이지 않다는 것은 특정 언어에서 사용하는 키워드나 예약어 등을 구분하지 않고 모양만으로 구분해서 생성함을 의미한다. 일반적인 어휘분석도구에서는 소스코드를 파싱하는데 불필요한 공백문자나 줄바꿈 등의 값은 필요한 수준의 정보(라인정보 등)만 얻은 후 버리지만, Python 과 같은 언어는 들여쓰기로 문법 구조를 파악하기 때문에 줄 바꿈이나 탭기호, 공백 문자의 길이까지 모두 토큰으로 생성한다. 표 1은 공통 어휘분석도구에서 생성 가능한 토큰의 예시이다. 각 토큰은 토큰의 종류를 담은 키워드와 부가적인 정보로 구성된다. 예를 들어, 주석(CMT)이나 문자열(DSLIT, SSLIT), 식별자(ID)는 주석의 내용, 문자열의 값, 식별자의 이름 등을 부가정보로 담는다. 더하기(+)나 쌍반점(;)과 같은 기호(symbol)는 별도의 부가정보 없이 토큰을 구성한다.

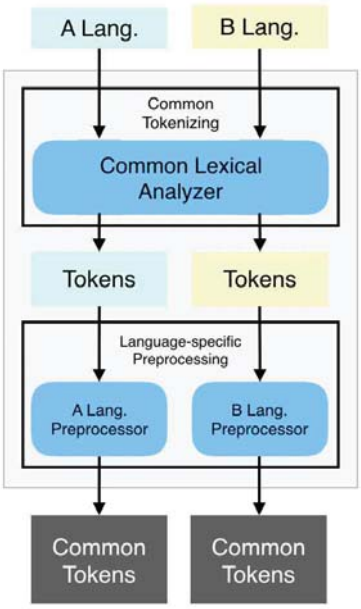


그림 2. 공통 토큰 생성 방법
Fig 2. Method for Common Token Generation

www.kci.go.kr

표 1. 공통 어휘분석도구가 생성하는 토큰 예시
Table 1. Tokens Generated by Common Lexical Analyzer

토큰종류	부가정보	설명
CMT	주석내용	주석
INC	지시문내용	지시문
IMP	import내용	import
INT	정수	정수
TAB	정수	들여쓰기
BLK	정수	빈 공간
DSLIT	문자열	쌍따옴표로 묶은 상수 문자열
SSLIT	문자열	따옴표로 묶은 상수 문자열
ID	문자열	변수, 키워드, 타입, 함수이름 등
NL	-	줄바꿈 문자 \n
PLUS	-	+
...	...	...

공통 어휘분석도구에서 생성한 토큰은 각 토큰이 담고 있는 의미가 명확하게 구분되지 않는다. 그래서 각 언어별로 전처리 과정(language-specific preprocessing)을 거친다. 이 과정에서 각 언어별 특성에 따라 토큰을 추가 혹은 삭제하거나 대체하는 작업을 수행한다.

가장 기본적인 작업은 식별자(ID) 토큰을 해당 언어에서 사용하는 키워드나 예약어로 바꾸는 작업이다. 대다수의 언어에서 사용하는 if나 while과 같은 키워드는 각각 식별자 ID“if”, ID“while”에서 별도의 키워드 토큰 IF, WHILE로 변환한다. Python과 같은 언어에서는 ID“elif”가 ELSE IF 의 두 키워드 토큰으로 변환되지만, Java에서는 변환 없이 ID“elif”로 남을 것이다.

경우에 따라선 없던 토큰을 추가로 생성할 필요가 있다. Python과 같은 언어는 여닫는 중괄호({, })를 사용하지 않고, 공백이나 탭 기호를 사용한 들여쓰기로 블록 구조를 구분한다. 그래서 Python언어의 전처리기는 공백이나 탭 기호, 줄바꿈 키워드에서 블록 구조를 알아낸 다음, 각 블록의 시작과 끝에 여닫는 중괄호를 추가해서 다른 언어와 같은 모양을 취하도록 한다.

마지막으로 필요한 작업을 마친 후, 줄 바꿈, 공백, 지시문이나 import 등(필요에 따라 주석까지) 정해진 토큰들을 삭제한다. 쌍반점(;)은 많은 프로그래밍언어에서 문장(statement)를 구분하는데 사용하지만 Python에서는 이를 사용하지 않는다. 블록구조를 알아내는 방법과 달리, 문장을 구분해내서 쌍반점을 추가하려면 구문분석수준의 작업이 필요하다. 이에 쌍반점 그 자체는 별다른 의미를 가지지 않으므로 다른 언어에서 일괄적으로 지우는 방법을 선택하였다. 표 2는 공통 토큰에서 무시하는 토큰이다.

표 2. 공통 토큰에서 빠진 토큰
Table 2. Unnecessary Tokens Eliminated in Common Tokens

토큰종류	설명
CMT	주석
NL	줄바꿈
TAP	들여쓰기
BLK	빈 공간
TYPE	타입
MOD	한정자
SEMICOL	세미콜론

표 3에 모든 전처리 과정을 마친 후에 남는 공통 토큰의 예시를 나열하였다. 공통 토큰은 공통 어휘분석도구에서 생성가능한 모든 토큰과 전처리 과정에서 추가되는 모든 토큰의 집합으로 표현할 수 있다.

공통 토큰 =
{ *공통 어휘분석도구에서 생성가능한 토큰* } +
{ *언어별 전처리기에서 추가/대체하는 토큰* } -
{ *언어별 전처리기에서 삭제되는 토큰* }

www.kci.go.kr

표 3. 공통 토큰의 예시 나열
Table 3. Examples of Common Tokens

토큰종류	부가정보	설명
INC	지시문내용	지시문
IMP	import내용	import
INT	정수	정수
DSLIT	문자열	쌍따옴표로 묶은 상수 문자열
SSLIT	문자열	따옴표로 묶은 상수 문자열
ID	문자열	변수, 함수이름 등
PLUS	-	+
IF	-	if 키워드
ELSE	-	else 키워드
...	...	...

언어별 전처리 과정을 마치고 생성한 공통 토큰에서는 식별자(ID)가 변수인지 함수인지까지는 분별하지 않는다. 이를 구별하는 건 구분분석 없이는 정확히 얻어 낼 수 없고, 토큰 기반의 유사성 검사에서는 구분이 되지 않더라도 주변의 정보(함수 호출의 경우 소괄호가 이어지는 등)로 자연스럽게 비교가 되기 때문에 추가 작업을 수행하지 않았다.

4. 실험

제안하는 방법을 확인하기 위해 Java와 Python으로 작성된 동일한 소스코드를 한국저작권위원회의 토큰기반 표절 검사 도구 exEyes로 실험하였다. 실험의 구성도는 그림 3과 같다.

먼저 softwareclones[13]에서 제공하는 연구 데이터 중 Java로 작성된 Eclipse Plugin[14]의 소스 일부를 사용하였다. 동일한 의미의 Python 코드 작성은 작성자 주관의 개입을 최소화하기 위해, Java로 작성된 코드를 Python으로 자동 변환해주는 오픈 소스 도구 java2python[15]으로 자동 생성하였다.

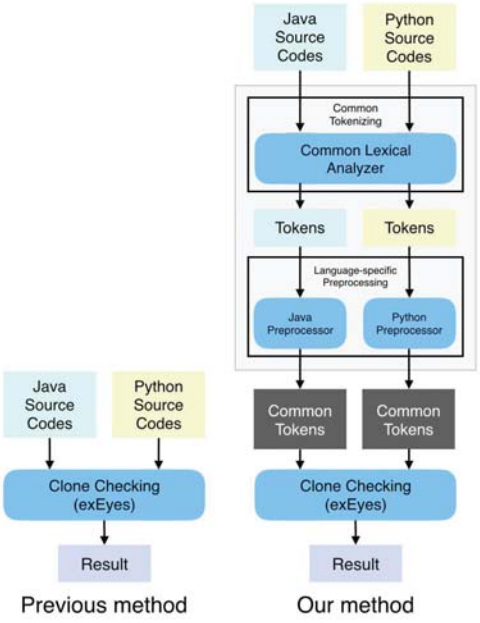


그림 3. exEyes를 이용한 실험 구성도
Fig 3. Test Configuration Using exEyes

제안하는 방법대로 그림 3의 우측과 같이, 먼저 Java와 Python 코드들을 공통 어휘분석도구로 토큰의 나열을 생성하였다. 이후 Java와 Python의 언어별 전처리 모듈로 공통 토큰의 나열을 생성한다. Java와 Python의 전처리 모듈은 다음과 같은 기준으로 작성하였다.

4.1 Java 전처리 모듈

Java의 문법정보는 Java SE 1.7을 기준으로 작업하였다.

4.1.1 대체

- 식별자(ID) 중 if, while 등 Java와 Python에서 공통으로 사용하는 키워드를 인지하여 토큰으로 대체하였다.
- 식별자(ID) 중 this, null은 고유의 키워드 토큰 THIS, NULL로 대체하였다.

www.kci.go.kr

4.1.2 추가

- 특별히 추가되는 토큰은 없다.

4.1.3 삭제

- 전처리 마지막 단계에서 표 2에 나열한 불필요한 토큰을 삭제하였다.

4.2 Python 전처리 모듈

Python의 문법정보는 Python 3.6.7 기준으로 작업하였다.

4.2.1 대체

- 입력으로 들어온 식별자(ID) 중 if, while 등 Java와 Python에서 공통으로 사용하는 키워드를 인지하여 토큰으로 대체하였다.
- Python에서만 키워드로 인지하는 and, or 등의 키워드는 연산자 키워드 AND, OR로 대체하였다 (Java에서는 그대로 식별자로 인지한다).
- 식별자(ID) 중 self, None은 Java의 this, null과 동등하게 인식하도록 고유의 키워드 토큰 THIS, NULL로 대체하였다.
- elif와 같이 Python에서만 사용하는 키워드는 Java와 의미가 같도록 연속된 두 공통 토큰 ELSE와 IF 로 대체하였다.

4.2.2 추가

- 공백, 탭기호, 줄바꿈 토큰을 기반으로 블록 구조를 파악하고 각 블록의 시작과 끝에 여닫는 중괄호({, })를 추가하였다. 또, 일반적으로 Python에서는 조건문과 반복문의 계산식에 여닫는 소괄호를 생략하므로, 이 역시 IF, WHILE 키워드와 여는 중괄호 앞에 여닫는 소괄호 ((,))를 추가하였다.

4.2.3 삭제

- 블록 구조 정보를 추가한 후, 반복문이나 조건문의 계산식 뒤에 이어지는 쌍점(:)은 불필요하므로 이를 삭제한다.
- Java와 마찬가지로 전처리 마지막 단계에서 표2에 나열한 불필요한 토큰을 삭제하였다.

표 4. exEyes의 기본 설정 값
Table 4. Default configuration of exEyes

설정항목	설정값
유효 동일/유사 블록 최소 크기	3줄
유사라인 판정 동일 토큰수 비율	80%
유사라인 판정 최소 토큰수	3개
유사라인 판정 최대 토큰비	2배
토크나이징 선행 여부	수행함

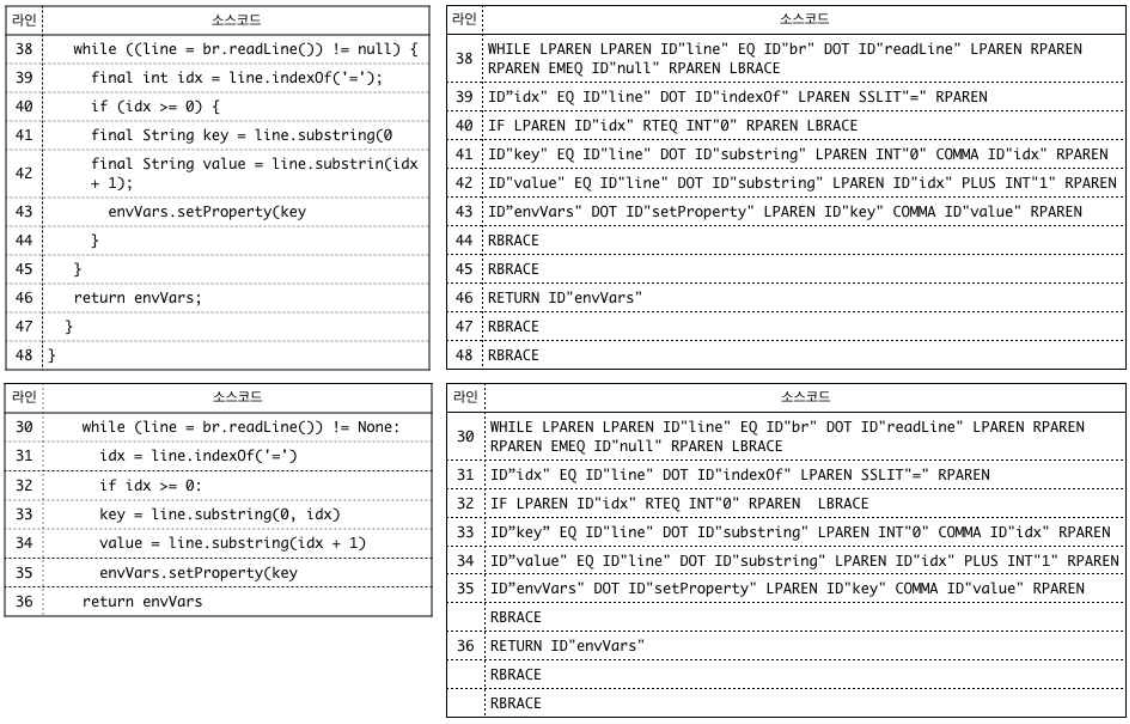
전처리 작업을 마치고 각 소스코드를 공통 토큰의 나열로 변환작업을 마치면, exEyes에서 제공하는 토큰의 목록을 입력으로 주는 기능을 사용해서 각 공통 토큰의 나열의 유사성을 검사하였다. 유사성 검사를 위한 설정 값은 표 4와 같이 기본값을 사용하여 진행하였다.

실험에 사용한 코드는 Eclipse Plugin의 Java 소스와 이를 Python으로 자동 변환한 소스 중, 5개를 선별하여 진행하였다. 선별은 소스코드의 길이가 다른 코드에 비해 길고 문법구조가 상대적으로 복잡한 것을 기준으로 삼았다. 선별한 파일의 목록과 라인 수는 표 5와 같다.

표 5. 실험대상 소스코드 목록 및 별칭
Table 5. Source Code Listings and Aliases

별칭	파일이름	라인 수
J1	AbstractProgramLauncher.java	135
J2	Activator.java	91
J3	FileLogger.java	75
J4	Utill.java	48
J5	Zipper.java	89
P1	AbstractProgramLauncher.py	110
P2	Activator.py	83
P3	FileLogger.py	65
P4	Utill.py	36
P5	Zipper.py	84

www.kci.go.kr



라인	소스코드
38	while ((line = br.readLine()) != null) {
39	final int idx = line.indexOf('=');
40	if (idx >= 0) {
41	final String key = line.substring(0
42	final String value = line.substrin(idx + 1);
43	envVars.setProperty(key
44	}
45	}
46	return envVars;
47	}
48	}

라인	소스코드
38	WHILE LPAREN LPAREN ID"line" EQ ID"br" DOT ID"readLine" LPAREN RPAREN RPAREN EMEQ ID"null" RPAREN LBRACE
39	ID"idx" EQ ID"line" DOT ID"indexOf" LPAREN SSLIT"=" RPAREN
40	IF LPAREN ID"idx" RTEQ INT"0" RPAREN LBRACE
41	ID"key" EQ ID"line" DOT ID"substring" LPAREN INT"0" COMMA ID"idx" RPAREN
42	ID"value" EQ ID"line" DOT ID"substring" LPAREN ID"idx" PLUS INT"1" RPAREN
43	ID"envVars" DOT ID"setProperty" LPAREN ID"key" COMMA ID"value" RPAREN
44	RBRACE
45	RBRACE
46	RETURN ID"envVars"
47	RBRACE
48	RBRACE

라인	소스코드
30	while (line = br.readLine()) != None:
31	idx = line.indexOf('=')
32	if idx >= 0:
33	key = line.substring(0, idx)
34	value = line.substring(idx + 1)
35	envVars.setProperty(key
36	return envVars

라인	소스코드
30	WHILE LPAREN LPAREN ID"line" EQ ID"br" DOT ID"readLine" LPAREN RPAREN RPAREN EMEQ ID"null" RPAREN LBRACE
31	ID"idx" EQ ID"line" DOT ID"indexOf" LPAREN SSLIT"=" RPAREN
32	IF LPAREN ID"idx" RTEQ INT"0" RPAREN LBRACE
33	ID"key" EQ ID"line" DOT ID"substring" LPAREN INT"0" COMMA ID"idx" RPAREN
34	ID"value" EQ ID"line" DOT ID"substring" LPAREN ID"idx" PLUS INT"1" RPAREN
35	ID"envVars" DOT ID"setProperty" LPAREN ID"key" COMMA ID"value" RPAREN
	RBRACE
36	RETURN ID"envVars"
	RBRACE
	RBRACE

그림 4. 공통 토큰 변환 사례 (J4,P4)
Fig 4. Example Results of Common Lexical Analysis (J4, P4)

표 6. 기존 방법과 제안 방법의 비교 실험 결과
Table 6. Experimental Results

구분	파일 유형	라인 수	파일 유형	라인 수	유사율	동일 라인	유사 라인
제안 방법 (공통 토큰)	J1	83	F1	81	57.83 %	24	24
	J2	34	F2	34	76.32 %	15	14
	J3	38	F3	37	76.47 %	15	11
	J4	32	P4	32	81.25 %	19	7
	J5	40	P5	36	77.50 %	15	16
기존 검사 방법	J1	118	P1	97	2.54 %	0	3
	J2	79	P2	75	3.80 %	0	3
	J3	75	P3	65	0 %	0	0
	J4	43	P4	34	6.98 %	0	3
	J5	84	P5	78	16.67 %	0	14

선별한 5개의 소스를 대상으로 유사성 검사를 수행한 결과는 표 6과 같다. 소스코드를 그대로 사용해서 exEyes로 비교한 기존검사방법에서는 J5, J2, J1, J3 4개의 파일에서 주석으로 작성된 부분만 유사하다고 결과가 나왔다. 유일하게 코드 부분이 같다고 나온 J4는 6.98%만 유사한 코드라는 결과를 보였다. 반면 공통 토큰의 나열로 변환 후 비교를 수행한 결과 최소 57.8%에서 최대 81.3%까지 더 높은 유사성 비율을 보였다.

그림 4는 공통된 토큰으로 비교 분석 한 결과에서 완전히 일치한다고 나온 일부분이다. 2개의 소스코드는 모양이 달라서 소스코드를 기존 방법에서는 다르다고 판정되었지만, 공통 토큰으로 변한 결과는 완전히 동일한 라인으로 구성됨을 확인할 수 있다. 이는 앞서 설명한 대로 블록의 구조를 파악하고 이를 식별하는 토큰을 추가한 것과, 비교에 불필요한 부분을 삭제하여 얻어진 결과이다.

www.kci.go.kr

5. 결론

본 연구는 서로 다른 언어로 작성되었지만 동일한 실행의미를 지닌 소스코드의 유사성 검사 성능을 높이기 위해 공통 토큰을 정의하고 이를 기반으로 유사성 검사를 수행하는 방안을 제안하였다. 소스코드를 언어와 상관없이 기본적인 토큰으로 구분한 다음, 각 언어별로 다른 모양을 지닌 토큰을 공통 토큰으로 변환하는 언어별 전처리 작업을 추가하였다. 언어별 전처리 작업은 단순히 토큰을 대체하는 수준이 아니라, 블록의 경계를 명확하게 표현해주는 방법 등을 더해서 구조적으로 다르게 판별될 수 있는 요소도 해결하였다.

토큰 기반 소스코드 표절 탐지 도구인 exEyes를 사용해서 Java와 Python으로 작성된 동일한 코드를 대상으로 실험했을 때, 기존의 방법에서 두 코드를 전혀 다른 코드라고 판정한 반면(0-16%), 제안한 방법에서는 유사한 코드(57.8%~81.%)로 판정하는 결과를 얻을 수 있었다. 유사하지 않게 판정되는 구문은 Java의 예외처리 구문이나 Python에서 변수의 선언 없이 바로 사용하는 등, 토큰수준의 유사성 검사에서는 분별이 불가능한 부분으로 파악되었다.

이런 약점은 토큰기반이 아닌 요약문법트리 기반의 유사성 검사에서 확실한 분별이 가능하므로, 제안하는 방법을 요약문법트리 기반의 유사성 검사에 적용하는 것을 추후과제로 남기고자 한다.

Acknowledgement

본 연구는 과학기술정보통신부 및 정보통신기술진흥센터의 SW중심대학지원사업의 연구결과로 수행되었음 (201800000001473)

참 고 문 헌

[1] Vislavski, Tijana, et al. "LICCA: A tool for cross-language clone detection." 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2018.

[2] Kraft, Nicholas A., Brandon W. Bonds, Randy K. Smith. "Cross-language Clone Detection." SEKE. 2008.

[3] Cheng, Xiao, et al. "Mining revision histories to detect cross-language clones without intermediates" Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. ACM, 2016.

[4] exEyes, http://www.copyright.or.kr, 2015.

[5] Baker, B., Manber, U., “Deducing similarities in java sources form byte codes”, In USENIX ATEC, pp.15-18, 1999.

[6] Johnson, J. H, “Identifying Redundancy in Source Code Using Finger-prints”, In CASCON, pp.171-183, 1993.

[7] Baker, B., “On Finding Duplication and Near-Duplication in Large Software Systems”, In WCRE, pp.86-95, 1995.

[8] Kamiy, T., Kusumoto, S., Inoue, K., “CCFinder: A multi-linguistic token based code clone detection system for large scale source code”, IEEE Trans. Software Engineering, Vol.28 No.7, Jul. 2002.

[9] Baxter, I., Pidgeon, C., Mehlich, M., “DMS: Program Transformation for Practical Scalabel Software Evolution”, In ICSE04, pp.625-634, 2004.

[10] Jiang, L., Misherghi, G., Su, Z., Glondu, S., “DECKARD: Scalable and Accurate Tree-based Detection of Code Clones”, In ICSE, pp.96-105, 2007.

[11] Komondoor, R., Horwitz, S., “Tool Demonstration: Finding Duplicated Code Using Program Dependences”, In ESOP, Vol. LNCS 2028, pp383-386, Apr. 2001.

www.kci.go.kr

[12] Liu, C., Chen, C., Han, J., Yu, P., "GPLAG: Detection of Software Plagiarism by Program Dependence Graph Analysis", In KDD pp.872-881, 2006.
[13] softwareclones,"http://www.softwareclones.org"
[14] Eclipse Plugin, "http://www.softwareclones.org/download/experiment/ex2clipse.tar.bz2"
[15] java2python, "https://github.com/natural/java2python"

저 자 소 개

홍성문(Sung-Moon Hong)

2012 위덕대학교 컴퓨터공학과 학사
2014 한양대학교 컴퓨터공학과 석사
2014-현재 : 한양대학교 대학원 컴퓨터공학과 박사과정
<주관심분야> 프로그래밍언어, 프로그램 분석, 소프트웨어 보안

김현하(Hyunha Kim)

2003 한양대학교 ERICA 전자컴퓨터공학부 학사
2005 한양대학교 컴퓨터공학과 석사
2013 한양대학교 컴퓨터공학과 박사
2013-2016 : 한양대학교 ERICA 컴퓨터공학과 리서치펠로우
2017-현재 : 소프토피아(주) CTO, 한양대학교 ERICA 소프트웨어학부 겸임교수
<주관심분야> 프로그래밍언어, 프로그램 분석, 소프트웨어 보안

이제형(Jaehyung Lee)

2015-현재 : 한양대학교 ERICA 소프트웨어학부 학사과정
<주관심분야> 프로그래밍언어, 프로그램 분석, 소프트웨어 보안

박성우(Sungwoo Park)

2016-현재 : 한양대학교 ERICA 소프트웨어학부 학사과정
<주관심분야> 프로그래밍언어, 알고리즘

www.kci.go.kr

저 자 소 개

모지환(Ji-Hwan Mo)

2017-현재 : 한양대학교 ERICA 소프트웨어학부 학사과정
<주관심분야> 프로그래밍언어, 프로그램 분석, 소프트웨어 보안

도경구(Kyung-Goo Doh)

1980 한양대학교 산업공학과 학사
1987 아이오와주립대학 컴퓨터과학 석사
1992 캔자스주립대학 컴퓨터과학 박사
1993-1995 일본 아이주 대학 교수
1995-현재 : 한양대학교 ERICA 소프트웨어학부 교수
<주관심분야> 프로그래밍언어, 프로그램 분석, 소프트웨어 보안, 소프트웨어 공학

www.kci.go.kr