논문 2012-2-2

CCR : 트리패턴 기반의 코드클론 탐지기

이효섭*, 도경구**

CCR : Tree-pattern based Code-clone Detector

Hyo-Sub Lee*, Kyung-Goo Doh**

요 약

본 연구에서는 트리패턴 기반으로 코드클론을 탐지하는 도구인 CCR(Code Clone Ransacker)를 제안하고 구현하였다. CCR은 프로그램 트리의 모든 하위트리 쌍을 비교하여 중복된 부분인 트리패턴을 찾고 동일한 모양의 패턴들을 하나로 묶어 프로그램에 존재하는 클론들을 샅샅이 탐지한다. 이때 이미 찾은 패턴 내부의 클론 패턴을 비교대상에서 제외하여 중복계산을 하지 않아 불필요한 예산을 최대한 줄인다. 실험으로 CCR의 성능을 평가한 결과, CCR의 정확성과 탐지성은 높다. 프로그램의 구조를 비교하는 기존의 트리패턴 기반의 코드클론 탐지 도구들의 정확성과 탐지성은 이미 좋은 것으로 알려져 있지만, CCR은 높은 정확성을 유지하면서 탐지성은 기존의 Asta보다는 최대 5배, CloneDigger보다는 약 1.9배 높다. 그리고 CCR이 찾은 코드클론은 기존의 코드클론 표본 집합체의 클론을 대부분 포함한다.

Abstract

This paper presents a tree-pattern based code-clone detector as CCR(Code Clone Ransacker) that finds all clusterd dulpicate pattern by comparing all pair of subtrees in the programs. The pattern included in its entirely in another pattern is ignored since only the largest duplicate patterns are interesed. Evaluation of CCR is high precision and recall. The previous tree-pattern based code-clone detectors are known to have good precision and recall because of comparing program structure. CCR is still high precision and the maximum 5 times higher recall than Asta and about 1.9 times than CloneDigger. The tool also include the majority of Bellon's reference corpus.

한글키워드 : 코드클론, 정확성(precision), 탐지성(recall), 표본 집합체, 트리 패턴

Ⅰ. 서 론

소프트웨어의 소스코드에서 중복되어 나타나는 동일한 모양의 코드 조각을 코드클론이라고 한다. 코드클론을 찾으면 방대하고 복잡한 기업 소프트웨어에서 소스코드에 잠재해 있는 코드클

*, ** 한양대학교 컴퓨터공학과
(email: {leehs, doh}@plasse.hanyang.ac.kr)
접수일자: 2012.12.20 수정완료: 2012.12.25.

※ 본 연구는 교육과학기술부/한국연구재단 우수연구센터 육성사업의 지원으로 수행되었음(과제번호 2012-0000469).

론을 찾아 소프트웨어의 품질관리와 유지보수를 할 수 있고 표절여부 감정평가나 리팩토링 등에 효과적으로 사용할 수 있다[1,2,10,11]. 그런데 코드를 사람의 눈으로 일일이 검사하여 코드클론을 찾아내는 작업은 검사자의 능력과 주어진 시간에 따라 결과가 차별적일 뿐만 아니라 경제적이지도 못하다. 따라서 이를 정밀하게 빨리 자동으로 찾는 도구가 필요하다.

코드클론을 자동으로 찾아주는 기존의 도구들은 다양한 탐지 기법을 이용해 구현한다. 도구 사용자는 기존의 도구 중에서 적용할 분야에 적합한 도구를 선택하기 위해 각 도구의 성능을 알아야하는데, 이때 성능을 비교 판단할 기준이 필요하다. 해당 도구의 우수성을 확인하거나 기존 도구들의 성능을 파악하기 위한 목적으로 비교할 때 공통적으로 사용하는 평가 항목에는 정확성(precision)과 탐지성(recall)이 있다[3,4,5,7,10,11]. 정확성은 도구에서 찾아낸 소스 코드가 모두 클론인지를 평가하는 것으로 도구에서 동일하다고 찾은 소스코드를 육안으로 직접 확인해서 서로 다른 코드를 클론으로 탐지한 것은 아닌지, 즉 오탐(false positives)이 없는지를 확인한다. 그리고 탐지성은 도구가 찾아내지 못하는 클론이 있는지 평가하는 것으로 도구에서 찾은 클론을 코드클론 표본 집합체(code clone reference corpus)와 비교해서 해당 도구가 탐지하지 못한 클론이 있는지, 즉 미탐(false negatives)이 없는지를 확인한다. 코드클론 표본 집합체는 여러 도구의 결과를 모아 표준이 될 만한 코드클론을 선택하여 구축한 클론의 모음이다[2,4]. 한 개의 도구에서 찾은 클론에는 오탐과 미탐이 있을 수 있어 정확한 표본을 구축할 수 없기 때문에 여러 도구에서 찾은 클론들의 수집이 필요하다.

본 연구에서는 우리의 알고리즘으로 구현한 트리패턴 기반의 코드클론 탐지기인 CCR(Code Clone Ransacker)을 소개한다. 그리고 CCR의 성능을 평가하고 기존 도구의 결과와 비교하기 위해 CCR의 정확성과 탐지성을 확인한다. 성능 평가 결과, CCR은 정확성과 탐지성이 매우 높다. 특히 CCR은 기존의 코드클론 탐지 도구인 Asta보다 최대 5배 그리고 CloneDigger보다 약 1.9배 탐지성이 높으면서 기존 코드클론 표본 집합체의 클론을 대부분 포함한다.

본 논문의 구성은 다음과 같다. 2절에서는 코드클론을 분류하는 네 가지 유형과 기존의 트리 기반 탐지 기법을 살펴본다. 3절에서는 트리패턴 기반으로 코드클론을 탐지하는 CCR의 알고리즘을 간단히 설명한다. 4절에서는 CCR의 탐지성과 정확성을 실험한 뒤 두 값의 관계에 대하여 알아본 뒤 기존의 도구와 탐지성을 비교하여 성능을 평가한다. 그리고 5절에서 결론과 추후 연구 기술로 마무리 한다.

2. 관련연구

코드 개발자들은 소스코드를 재사용하기 위해 원형 소스코드의 복제를 시도하지만 클론으로 보이지 않으려고 원본과 의미적으로는 동일하거나 유사하면서 모양새는 다르게 변형하려고 한다. 일반적으로 많이 시도하는 변경 방법들을 분류하면 다음의 네 가지로 유형을 나눌 수 있다[10,11].

- Type 1 : 공백과 주석, 레이아웃을 제외하고 동일
- Type 2 : 식별자, 상수, 타입을 제외하고 구조적/ 외형적으로 동일
- Type 3 : 명령어의 변경 또는 삽입과 삭제를 제외하고 동일
- Type 4 : 서로 다른 구문을 사용하지만 기능은 동일

위의 네 가지 코드클론 유형에서 하위 유형은 상위 유형의 모든 변경 방법을 포함한다. 예를 들어 type 3은 명령어의 변경뿐만 아니라 상위 유형인 type 1의 클론 조건인 공백과 주석, 레이

아웃의 변경 그리고 type 2의 식별자, 상수, 타입 변경까지 모두 포함한다.

다양한 코드클론 유형으로 변경한 소스코드를 클론으로 찾기 위해 기존의 연구에서는 프로그램을 비교하는 단위와 구조에 따라 문자열기반, 토큰기반, 트리기반, 계량치기반, 의존그래프기반, 혼합 방식으로 탐지 기법을 나눈다[10]. 여섯 가지 탐지 기법에서 계량치기반과 의존그래프기반 방식은 코드가 달라도 계량치나 그래프가 동일하면 클론으로 탐지하기 때문에 오탐이 생길 수 있다. 따라서 순수하게 프로그램의 모양만을 기준으로 비교하는 문자열기반, 토큰기반, 트리기반 방식이 적절하다. 그러나 식별자나 빈칸 삽입과 같은 거의 의미 없는 차이를 구별하는 문자열기반 방식은 효율성이 떨어지고, 구조를 전혀 고려하지 않는 토큰기반 방식은 원천적으로 오탐을 찾아내기 쉽다. 그러므로 코드의 구문구조 정보가 포함되어 있는 추상구문트리(abstract syntax tree)를 비교하는 트리기반 방식이 코드클론을 찾는데 가장 적합하다고 할 수 있다.

트리기반은 하위트리를 비교 단위로 하여 구문구조의 모양을 기준으로 코드클론을 찾는 방법으로, 기존의 탐지 기법보다는 오탐과 미탐이 적다. 그러나 코드클론을 탐지하는데 걸리는 시간이 크다. 이러한 시간적 제약을 극복하기 위하여 기존 연구에서는 특정 응용 분야에 적합한 해쉬함수를 고안하여 비교하거나[1] 구조정보를 특질벡터(characteristic vector)로 요약한 다음 비교하는 방식[6]을 제안하였다. 이와 같은 방식은 훨씬 빠른 시간 안에 클론을 찾아내는 장점이 있지만, 단점으로는 오탐이 있어 추가적인 검증과정이 필요하다. 따라서 코드클론을 보다 정확히 찾아내기 위해 CCR을 포함한 트리기반의 일부 도구에서는 anti-unification 알고리즘을 이용하여 도구를 구현하였다. Evans-Fraser의 Asta[5]는 트리의 중복 비교를 피하기 위해 동적 계획법으로 클론 패턴을 탐지한다. 그러나 Asta는 상향식 방법으로 작은 클론을 모으는 과정에서 가장 큰 클론을 찾기 때문에 놓치는 클론이 있으며 겹침이 있는 소스코드는 제거하기 때문에 크기가 큰 클론도 일부 탐지하지 못한다. Bulychev-Minea의 CloneDigger[3]는 클론의 단위를 문장이나 문장의 나열로 제한하기 때문에 탐지하는 클론도 제한적이다. 이외에 Li-Tohmpson의 연구[9]에서는 suffix-tree를 이용해 검출한 클론이 정확한지 확인하기 위해서 anti-unification 알고리즘을 사용해 다시 한 번 클론 여부를 검토한다.

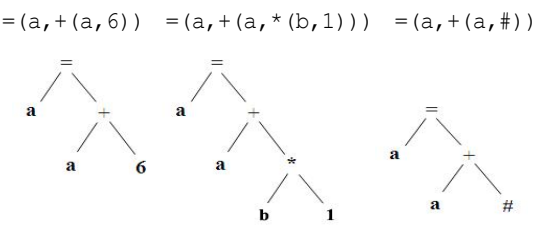
3. 트리패턴 기반의 코드클론 탐지

이 절에서는 트리패턴을 이용해 프로그램의 코드클론을 찾는 새로운 방법을 기술한다. 이 알고리즘은 두 트리를 입력으로 받아 가능한 모든 하위트리의 쌍을 비교하여 중복 부분인 트리패턴을 찾고 이들을 병합하여 프로그램에 존재하는 클론들을 찾는다.

3.1 중복 패턴과 클러스터 패턴

트리 조각(tree slice)은 트리에서 서로 연결된 노드들의 덩어리이다. 주어진 트리에서 정확히 일치하는 트리 조각을 모으면(MSG ; the most specific generalization) 루트 노드를 공유하는 연결된 가장 큰 공통된 부분인 최대 공통 트리 조각을 찾는다. 예를 들어 주어진 트리는 그림 1의 (a)와 같이 `=(a,+(a,6))`와 `=(a,+(a,*(b,1)))`이다. 그리고 두 트리의 MSG는 `=(a,+(a,#))`이다. 이 트리에는 구멍(hole)이라 부르며 문자 '#'으로 표시하는 특별한 노드가 있는데, 원본 트리에서 서로 다른 부분인 하위트리, 즉 `6`과 `*(b,1)`를 대체한 것이다. 이와 같은 방법으로 생성한 두 트리의 MSG는 그림 1의 (b)이며 트

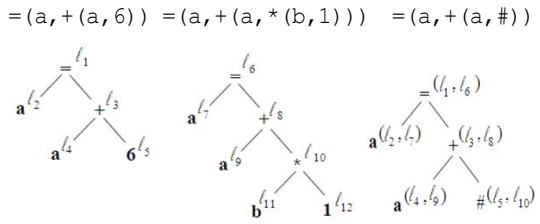
리패턴(tree pattern)이라고 한다. 트리패턴은 0개 이상의 구멍을 포함한다.



(a) 두 트리 (b) 트리패턴

그림 1. 두 트리의 트리 패턴

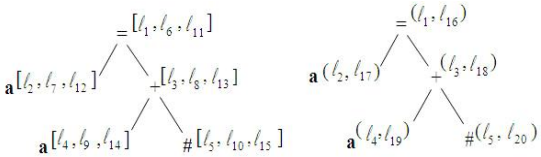
프로그램 트리(program tree)는 프로그램을 구문 분석하여 핵심구문트리(AST; abstract syntax tree)로 변환한 것이다. 프로그램 트리의 노드에는 프로그램 텍스트의 위치정보(라인과 컬럼 정보)를 저장한 꼬리표가 붙는다. 그림 2의 (a)는 그림 1의 (a)에 꼬리표가 붙은 프로그램 트리이다. 그림 2의 (b)는 (a)의 두 프로그램 트리의 중복 패턴(duplicate pattern)으로 두 프로그램 트리에서 MSG로 탐지한 두 트리의 뼈대에 원본 노드의 위치 정보를 쌍으로 묶어서 각 노드에 꼬리표를 달았다.



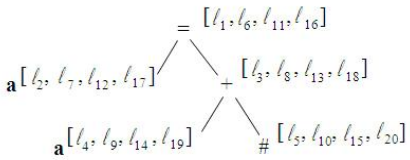
(a) 두 프로그램 트리 (b) 중복 패턴

그림 2. 두 프로그램 트리의 중복 패턴

동일한 모양의 중복 패턴은 하나로 묶으면 병합된 중복 패턴(clustered duplicate pattern)이다. 병합된 중복 패턴은 클러스터 패턴으로도 부른다. 각 노드의 위치 정보는 리스트로 표현해서 클러스터 패턴의 노드에 꼬리표를 단다. 그림 3은 동일한 모양을 가진 클러스터 패턴과 중복 패턴을 합병하여 하나의 클러스터 패턴으로 만든 것이다. 그림 3의 (a)에서 클러스터 패턴과 중복 패턴의 모양은 동일하다. 그러나 노드의 꼬리표는 크기와 내용이 다르기 때문에 하나의 리스트로 합병해야 한다. 이때 두 패턴의 루트 노드에서는 꼬리표의 내용 중 1 이 중복되므로 중복된 정보를 제외한 뒤 꼬리표의 정보를 병합한다. 나머지 모든 노드의 꼬리표도 동일한 위치 정보가 있다면 제외하고 병합한다.



(a) 클러스터 패턴과 중복 패턴의 병합



(b) 클러스터 패턴 (병합된 중복 패턴)

그림 3. 동일한 모양의 중복 패턴과 클러스터 패턴의 병합

3.2 코드클론 자동 탐지 알고리즘

두 개의 트리를 입력받아서 클러스터 패턴을 내주는 이 알고리즘은 크게 두 단계로 나눈다[9].

첫 번째 단계에서는 MSG로 프로그램의 모든 중복 패턴을 찾는다. 우선 주어진 두 프로그램의 프로그램 트리에서 각 노드를 루트 노드로 하는 가능한 모든 트리의 쌍을 비교하여 동일한 모양새를 가지는 연결된 가장 큰 공통된 패턴인 중복

패턴을 찾는다. 트리는 루트 노드부터 비교해서 두 노드가 동일하면 깊이 우선으로 순회하면서 패턴을 구축하지만 서로 다른 노드는 구멍이 되어 더 이상 하위트리를 비교하지 않는다. 이와 같이 구멍이 있는 트리패턴을 작성하는 방법을 anti-unification이라고 한다. 그리고 이때 비교하는 프로그램 트리의 모든 노드는 작업 테이블에 표시하여 이미 찾아낸 패턴 내부의 작은 패턴은 비교 대상에서 제외한다. 따라서 이미 비교한 두 노드는 중복 계산을 하지 않으므로 최대한 불필요한 계산을 줄일 수 있다.

두 번째 단계에서는 프로그램의 여러 곳에서 찾은 동일한 모양의 중복 패턴을 하나로 묶어 클러스터 패턴을 만든다. 동일한 모양의 중복 패턴들을 병합하기 때문에 각 노드의 위치정보 리스트 정보인 꼬리표도 중복된 내용은 제거한 뒤 병합한다. 따라서 두 번째 단계의 실질적인 결과는 클론 패턴의 클래스인 소스코드의 조각들이다.

4. 실험과 평가

이 절에서는 제 3절의 트리 패턴 기반의 클론 탐지 알고리즘으로 구현한 우리 도구 CCR(Code Clone Ransacker)을 소개한다. 먼저 실험 대상인 네 개의 Java 응용 프로그램을 설명하고 CCR의 파라미터 사용법과 특징을 기술한다. 그리고 CCR의 성능을 평가하기 위해 탐지성(recall)과 정확성(precision)을 실험을 통해 분석하고 다른 도구의 성능과 비교 평가한다.

4.1 실험대상

CCR의 결과를 평가하기 위해 네 개 Java 응용 프로그램의 소스코드를 비교 실험 대상으로 정했다. 이 프로그램들은 CCR과 결과를 비교할 Asta[5]와 Clone Digger[3] 그리고 코드클론 표본 집합체를 작성한 연구[2]에서도 동일하게 사용한 것이다.

표 1은 비교 대상인 네 개 응용 프로그램의 크기(MB)와 파일 수 그리고 줄 수이다. 응용 프로그램은 크기가 약 0.7-8.3 MB이며 파일의 개수가 대략 100-700개이고 1만 4천-2십만 줄로 구성된 소스코드이다.

표 1. 비교대상인 응용 프로그램들

응용 프로그램	크기 (MB)	파일 수	줄 수
netbeans-javadoc	0.69	97	14,301
eclipse-ant	1.35	149	29,880
eclipse-jdtcore	6.37	687	135,675
j2sdk1.4.0-javax-swing	8.32	533	202,943

4.2 도구의 구현과 실험 방법

CCR은 Java로 작성된 프로그램의 코드클론을 찾기 위해 트리패턴을 이용하여 자동으로 클론을 탐지하는 도구이다. 이 도구는 프로그램의 구문 분석을 위해 Objective Caml로 작성된 파서 Joust 0.8[1]을 이용하였고 Objective Caml 3.09와 Python 2.5.1으로 구현하였다.

CCR에서 도구 사용자는 사용자 요구사항을 만족하는 클론을 찾기 위해 도구에서 필요로 하는 세 개의 패턴 속성 값을 입력한다. 첫 번째 파라미터인 `MinNode`는 크기가 너무 작아서 클론으로 의미가 없는 패턴을 배제하기 위해 허용하는 최소 토큰의 개수이다. 두 번째 파라미터 `MaxHole`은 비교하는 두 프로그램 코드의 서로 다른 부분의 개수로, 하나의 패턴에서 최대 수용 가능한 구멍의 개수이다. 마지막으로 파라미터 `HoleMassLimit`는 트리패턴에서 구멍의 크기가 너무 크면 클론이라고 보기 어렵기 때문에 패턴의 각 구멍이 허용하는 최대 구멍 크기이다.

1) http://www.cs.cmu.edu/~ecc/joust.tar.gz

도구 CCR의 기본값으로 정한 파라미터 값은 MinNode=20, MaxHole=15, HoleMassLimit=5 이다. 이 값은 코드클론을 탐지하는 사용도구인 CloneDR의 기본 파라미터 값[2]으로 실험한 결과와 유사한 환경을 제공한다. 기존의 클론탐지 도구[4,14]에서도 CloneDR의 기본 파라미터 값을 기준으로 유사한 환경을 만들어 파라미터 기본값을 정해 실험한 뒤 성능을 평가한다. 엄밀히 말해 CloneDR의 파라미터와 CCR의 파라미터는 의미와 항목이 일치하지 않는다. 따라서 여러 번의 모의실험을 통해 CCR의 파라미터 기본값이 CloneDR의 기본값과 거의 동일한 조건의 실험이 될 수 있도록 값을 조정하였다.

CCR의 세 개의 파라미터에서 MaxHole의 값이 0이면 CCR은 완벽하게 동일한 클론을 탐지한다. 그러나 클론 유형이 type 2 또는 type 3인 클론을 찾기 위해서는 MaxHole의 값이 0보다 커야 하므로 서로 다른 부분의 개수인 MaxHole의 값이 15이면 실험의 초기값으로 충분하다. CCR의 HoleMassLimit 기본값 5는 패턴의 각 구멍 크기가 5보다 작거나 같은 클론을 찾는다.

4.3 탐지성

탐지성은 CCR이 주어진 소스코드 내의 클론을 빠짐없이 탐지하는가를 확인하는 성능 평가 항목이다. 탐지성은 해당 도구의 결과만을 분석하여 성능을 평가할 수 없기 때문에 절대클론의 모음인 코드클론 표본 집합체와 비교해야 한다. CCR의 패턴이 Bellon의 코드클론 표본 집합체의 클론 쌍과 얼마나 일치하는지 알아보기 위해 직접 하나하나 눈으로 클론을 확인하여 비교하였다. 그러나 프로그램의 크기가 너무 커서 수작업 비교가 현실적으로 불가능한 두 개의 응용 프로그램 eclipse-jdtcore와 j2sdk1.4.0-javax-swing은 전체의 5%만 비교대상으로 선정하였다. 비교대상 집합체 선정은 Bellon의 코드클론 표본 집합체의 클론 유형 type 1, 2, 3과 동일한 비율을 유지하면서 일련번호가 큰 5%를 선택하였다.

탐지성 실험에서는 CCR의 파라미터 기본값으로 찾은 클론을 Bellon의 표본 집합체와 비교하지만, 이 집합체를 최대 얼마나 더 찾는지 보기 위해 일부 파라미터의 값을 변경하여 세 번의 실험을 추가한다[3]. 우선 파라미터 MinNode의 값은 20으로 고정하고 클론의 줄 수도 최소 6줄로 제한하였다. 기존의 도구들도 6줄 이상의 소스코드만을 클론으로 인정하기 때문에[2], CCR은 자동으로 생성한 클론 패턴 클래스에서 5줄보다 작거나 같은 크기의 소스코드를 모두 제거하였다. 그리고 파라미터인 MaxHole과 HoleMassLimt의 값을 차례로 변경하여 실험하였다. 파라미터 MaxHole과 HoleMassLimt의 값 선정은 코드클론 유형 type 1, 2, 3을 코드클론 표본 집합체의 클론 유형과 동일하게 유지하되 20%를 임의 선정하여 특징을 분석한 예비실험을 통해 정했다. 분석결과 가장 많이 변경한 유형은 type 2의 표현식 변경이다. 따라서 추가 실험에서는 도구의 기본값에서 가장 먼저 표현식 변경을 수용하는 파라미터 HoleMassLimit의 값을 5에서 10으로 변경하였다. 그러나 예비실험은 세 번의 추가실험에서 사용할 파라미터의 값을 선정하기 위한 것이므로 실험 2-4의 결과와는 다를 수 있다.

표 2는 각 실험에서 찾은 CCR의 탐지성이다. CCR의 탐지성은 임계치가 0.7이상인 good-value와 동일하다.

2) CloneDR의 트리기반 방식의 대표적인 상용도구이다. CloneDR의 기본 파라미터 값은 similarity threshold=95%, Maximum parameters=6, minimum clone mass=4, number of characters per node=16, initial starting depth=2이다.

3) CCR의 탐지성에 대한 자세한 실험결과는 다음 사이트에서 확인할 수 있다 ;
http://plasse.hanyang.ac.kr/ccr

표 2. CCR의 탐지성

응용 프로그램	Bellon의 코드클론 표본 집합체	탐지성			
		실험 1	실험 2	실험 3	실험 4
netbeans-javadoc	55	36 (65.5%)	37 (67.3%)	40 (72.7%)	42 (76.4%)
eclipse-ant	30	28 (93.3%)	28 (93.3%)	30 (100.0%)	30 (100.0%)
eclipse-jdtcore	67	62 (92.5%)	62 (92.5%)	62 (92.5%)	62 (92.5%)
j2sdk1.4.0-javax-swing	38	36 (94.7%)	37 (97.4%)	37 (97.4%)	38 (100.0%)
전체	190	162 (85.3%)	164 (86.3%)	169 (88.9%)	172 (90.5%)

● **실험 1 : 파라미터 기본값으로 찾은 탐지성**

실험 1은 CCR의 파라미터 기본값인 MinNode= 20, MaxHole=15, HoleMassLimit=5로 탐지한 클론 패턴 클래스와 Bellon의 코드클론 표본 집합체를 비교하여 탐지성을 확인한다. 표 2에서 비교 대상인 코드클론 표본 집합체의 개수 190개중 CCR은 162개의 클론 쌍을 탐지하므로 실험 1의 평균 탐지성은 85.3%이다.

그림 4는 CCR의 파라미터 기본값으로 실험한 실험 1에서 탐지한 클론들을 코드클론 유형별 비율로 표시한 것이다. CCR은 네 개 응용 프로그램의 클론 유형 type 1을 모두 탐지한다. 그리고 응용 프로그램 eclipse-jdtcore와 j2sdk1.4.0-javax- swing에서 클론 유형 type 2인 클론 쌍과 응용 프로그램 eclipse-ant의 클론 유형 type 3인 클론 쌍도 모두 탐지한다. 그러나 전체적으로는 클론 유형 type 2의 6개와 type 3의 22개, 총 28개의 클론 쌍은 탐지하지 못한다.

● **실험 2 : 파라미터 HoleMassLimit=10으로 찾은 탐지성**

실험 2는 CCR의 변경된 파라미터의 값으로 찾은 코드 클론의 탐지성을 확인하기 위해 실험 1의 파라미터 중에서 HoleMassLimit의 값을 5에서 10으로 변경하고 그 외의 파라미터 값은 그대로 유지하였다. 파라미터 HoleMassLimit의 값을 5에서 10으로 변경하면 식별자의 이름을 바꾸거나 함수의 매개변수 개수를 변경한 소스코드를 클론으로 찾는다. 표 2의 실험 2와 같이 CCR은 164개의 클론 쌍을 탐지하며 실험 1보다 두 개의 클론을 더 탐지해 평균 탐지성이 86.3%이다.

그림 5는 파라미터 HoleMassLimit의 값을 10으로 변경해서 탐지한 클론들의 코드클론 유형별 비율이다. 추가 탐지한 두 개의 클론 쌍은 응용 프로그램 netbeans-javadoc의 클론 유형 type 2와 응용 프로그램 j2sdk1.4.0-javax-swing의 클론 유형 type 3에서 각각 한 개다.

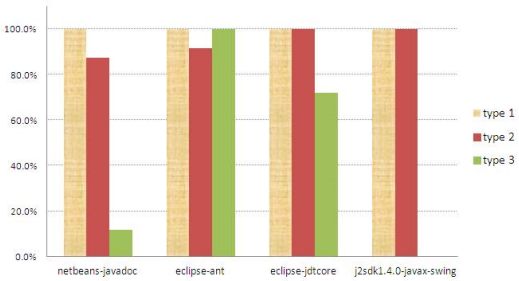


그림 4. 파라미터 기본값으로 실험한 코드클론 유형별 탐지성

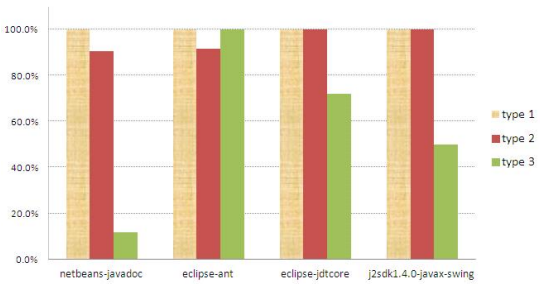


그림 5. 파라미터 HoleMassLimit=10으로 실험한 코드클론 유형별 탐지성

● **실험 3 : 파라미터 MaxHole=25로 찾은 탐지성**

실험 3은 CCR의 세 개의 파라미터에서 MaxHole의 값을 15에서 25로 변경하고 그 외의 파라미터는 실험 2의 값으로 유지하였다. 파라미터 MaxHole의 값을 15에서 25로 변경하면 실험 2보다 구멍의 개수가 더 많은 트리패턴을 클론으로 찾는다. 따라서 실험 3에서는 프로그램의 구조는 거의 동일하지만 식별자와 매개변수의 변경이 비교적 많은 소스코드를 클론으로 추가 탐지한다. 표 2의 실험 3에서 CCR의 평균 탐지성은 88.9%이며 실험 2보다 다섯 개의 클론 쌍을 더 탐지해 총 169개의 클론 쌍을 찾는다. 이때 추가 탐지한 클론 쌍의 클론 유형은 모두 type 2이다. 그림 6은 파라미터 MaxHole의 값을 25로 변경해서 탐지한 클론들의 코드클론 유형별 비율이다. 이번 실험에서 CCR은 Bellon의 코드클론 표본 집합체의 클론 유형 type 1과 type 2인 클론 쌍을 모두 탐지한다. 즉 이번 실험에서는 표본 집합체의 모든 응용 프로그램에서 미탐지하는 type 1과 type 2인 클론 쌍이 없다. 그러나 클론 유형 type 3의 탐지성은 실험 2의 결과와 동일하다. 클론 유형이 type 3인 클론의 비율은 응용 프로그램 eclipse-ant에서 100.0%이지만 netbeans-javadoc에서 11.8%이고 eclipse-jdtcore에서 72.2%이며 j2sdk1.4.0-javax-swing에서 50%이다.

● **실험 4 : 파라미터 HoleMassLimit=25로 찾은 탐지성**

실험 4는 CCR의 세 개의 파라미터에서 HoleMassLimit의 값을 10에서 25로 변경하고 그 외의 파라미터는 실험 3에서 사용한 값을 유지하였다. 파라미터 HoleMassLimit의 값을 25로 변경하면 간단한 문장을 삽입하거나 기존의 짧은 문장을 전혀 다른 모양으로 변경한 소스코드도 클론 패턴으로 탐지한다. 즉 프로그램의 구멍의 크기가 비교적 큰 프로그램의 문장 변경도 CCR은 트리 패턴으로 탐지한다. 표 2의 실험 4에서 CCR의 평균 탐지성은 90.5%이다. 이 실험은 실험 3과 비교해 Bellon의 코드클론 표본 집합체에서 세 개의 클론 쌍을 더 탐지해 총 172개의 클론 쌍을 탐지하며 추가 탐지한 코드클론의 유형은 모두 type 3이다.

그림 7은 파라미터 HoleMassLimit의 값을 25로 변경해서 탐지한 클론들의 코드클론 유형별 비율이다. 두 개의 응용 프로그램 eclipse-ant와 j2sdk1.4.0-javax-swing의 탐지성은 100.0%로 CCR은 Bellon의 코드클론 표본 집합체의 클론 쌍을 모두 탐지한다. 그러나 응용 프로그램 netbeans-javadoc에서는 클론 유형 type 3인 13개의 표본 집합체를 찾지 못해 탐지성이 23.5%이며, eclipse-jdtcore는 type 3인 5개의 클론 쌍을 탐지하지 못하기 때문에 탐지성이 23.5%이다.

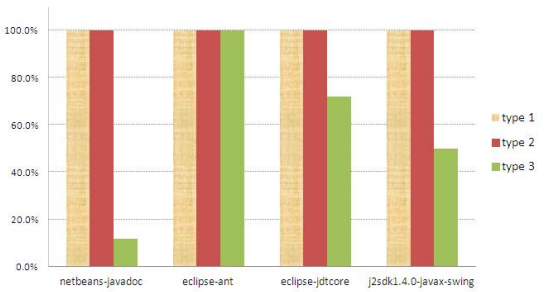


그림 6. 파라미터 MaxHole=25로 실험한 코드클론 유형별 탐지성

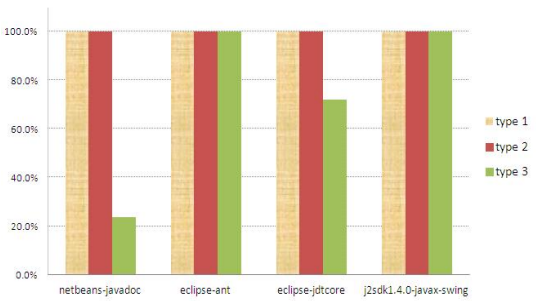


그림 7. 파라미터 HoleMassLimit=25로 실험한 코드클론 유형별 탐지성

4.4 정확성

정확성은 CCR이 찾아낸 코드클론이 실제로도 정확한지, 즉 오탐이 어느 정도인가를 확인하는 성능 평가 항목이다. 따라서 CCR의 정확성을 알아내기 위해 도구에서 탐지한 클러스터 패턴의 모든 소스코드를 육안으로 하나하나 검사하여 클론인지 확인하였다.

네 번의 실험에서 탐지하는 클러스터 패턴의 개수는 표 3과 같다. 실험 1과 실험 4에서 탐지한 클러스터 패턴의 개수는 응용 프로그램에 따라 1.5 - 3.3배의 차이가 난다.

표 3. CCR에서 찾아낸 클론 클래스의 개수

실험 번호		실험 1	실험 2	실험 3	실험 4
파라미터	MaxHole	15	15	25	25
	HoleMassLimit	5	10	10	25
netbeans-javadoc		139	151	165	209
eclipse-ant		152	176	211	341
eclipse-jdtcore		1913	2074	2671	3846
j2sdk1.4.0-javax-swing		1061	1380	2092	3452
전체		3265	3781	5149	7848

CCR은 소스코드의 요약트리에서 모양을 비교하여 공통부분인 트리패턴을 찾고 동일한 트리패턴을 병합해서 클러스터 패턴을 작성하기 때문에 CCR이 탐지한 클러스터 패턴의 해당 소스코드들은 모양새가 정확히 동일하다. 그러나 소스코드의 모양새가 동일하더라도 패턴의 구멍 개수가 많거나 구멍의 크기가 크면 육안으로 클론을 판정하기 어려울 정도로 서로 다를 수 있다. 엄밀히 말해 비교하는 두 소스코드를 코드클론이라고 결정하는 것은 매우 주관적인 판단이 필요하다. 다시 말해, CCR은 구조가 일치하는 패턴을 찾지만 클론이라고 하기 어려운 패턴도 일부 있기 때문에 사용자는 패턴에 해당하는 소스코드들이 클론인지 여부를 확인해야 한다. 이와 같이 뼈대만 동일한 클론은 기존의 연구에서는 의미 없는 클론으로 분류되기도 한다[11]. 표 4는 자동으로 탐지한 클러스터 패턴인 표 3의 모든 소스코드를 직접 눈으로 확인하여 찾은 오탐 클론의 개수이다. 파라미터의 값이 증가할수록 오탐의 비율도 증가하기 때문에 비교해야할 클러스터 패턴의 수가 너무 많은 일부 응용 프로그램은 육안 확인을 생략하였다[4]. 실험 2에서 찾은 오탐 클론의 개수는 실험 1과 비교해서 응용 프로그램 netbeans-javadoc에서 1.6배(최소) 그리고 응용 프로그램 eclipse-jdtcore에서 4.0배(최대) 더 많다.

표 4. CCR에서 찾아낸 오탐 클론의 개수

실험 번호		실험 1	실험 2	실험 3	실험 4
파라미터	MaxHole	15	15	25	25
	HoleMassLimit	5	10	10	25
netbeans-javadoc		5	8	17	47
eclipse-ant		4	14	35	153
eclipse-jdtcore		34	136	-	-
j2sdk1.4.0-javax-swing		41	102	-	-
전체		84	260	52	200

그림 8은 실험을 진행하면서 변하는 CCR의 오탐 클론 비율이다. 실험 1과 2에서는 오탐의 비율이 1.8 - 3.6%에서 5.3 - 8.0%으로 변화율이

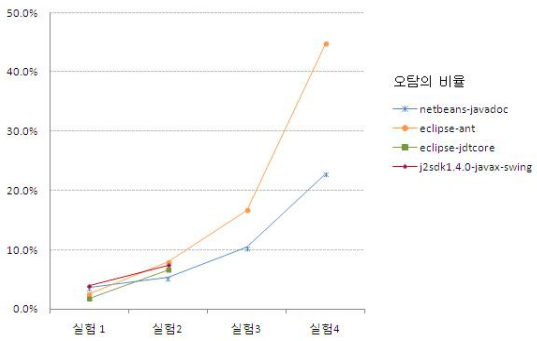


그림 8. CCR의 오탐 클론 비율

4) 탐지한 클론 개수가 많아 육안학인이 어려운 응용 프로그램 eclipse-jdtcore와 j2sdk1.4.0-javax-swing의 실험 3,4는 정확성 실험에서 제외하였다.

낮지만 실험 3과 실험 4의 오탐 비율은 변화율이 크기 때문에 꺽은선의 기울기가 가파르다.

CCR에서 육안으로 찾은 오탐 클론을 분류하면 다음과 같다.

① 몸체가 없는 추상 메소드의 연속적인 나열
② 몸체의 구조가 유사한 메소드의 연속적인 나열
③ 특정 문장의 연속적 나열 (catch, case, if 문)
④ 변수 선언문의 연속적인 나열

위 리스트에서 오탐 클론의 비율이 가장 높은 항목은 ②인 몸체가 구조가 유사한 메소드의 연속적인 나열이다. 항목 ②의 비율은 실험 1에서 65.5%, 실험 2에서 71.5%, 실험 3에서 53.8% 그리고 실험 4에서는 69.0%이다. 표 5는 오탐 클론 ②의 예제로 실험 3에서 탐지하였다. 2줄 또는 1줄의 매우 간단한 몸체를 가진 세 개의 메소드가 연속적으로 있는데, 소스코드의 전체 모양새와 일부 키워드 public과 void는 동일하지만 그 외의 소스코드의 타입과 식별자는 모두 다르다.

표 5. 몸체가 있고 유사한 구조를 가진 메소드의 연속적인 나열

CCR's ID #163 (netbeans-javadoc)

Clone 1 : File Name - DocFSoffsetEditor.java (lines 50-57)

```
50  public void addPropertyChangeListener (PropertyChangeListener l) {
51      super.addPropertyChangeListener(l);
52      supp.addPropertyChangeListener (l); }
53  public void removePropertyChangeListener (PropertyChangeListener l) {
54      super.removePropertyChangeListener(l);
55      supp.removePropertyChangeListener (l); }
56  public boolean supportsCustomEditor () {
57      return true; }
```

Clone 2 : File Name - RootDocImplWrapper.java (lines 152-153)

```
152  public void printNotice(String msg) {
153      out.println ( msg );
154      out.flush();
155  } //Res.printNotice(msg);
156  public void printNotice (SourcePosition pos, String msg) {
157      out.println ( msg );
158      out.flush();
159  }//Res.printNotice(msg);
160  public SourcePosition position() {
161      return null; } }
```

4.5 탐지성과 정확성의 관계

파라미터의 값을 변경해서 실험을 진행할수록 탐지성은 높아지고 정확성은 낮아진다. 앞의 4.3에서는 CCR의 탐지성을 확인하기 위해 가장 먼저 CCR의 파라미터 기본값으로 실험하였다. 그러나 CCR의 파라미터 기본값으로는 Bellon의 코드클론 표본 집합체의 모든 클론 쌍을 탐지할 수 없었다. 따라서 CCR의 파라미터 값 조절로 최대한 찾을 수 있는 가능한 모든 클론 쌍을 탐지하기 위해 실험을 추가하였다. 실험 1에서는 CCR의 평균 탐지성이 85.3%이지만 두 개의 파라미터 값이 커질수록 탐지성은 높아져 실험 4에서는 평균 탐지성이 90.5%이다. 그리고 4.4에서는 CCR에서 탐지한 클러스터 패턴의 클래스에서 정확성을 확인하기 위해 오탐 클론의 개수와 비율을 실험 1부터 실험 4까지 육안으로 확인하였다. CCR의 정확성은 실험 1에서 96.1 - 98.2%로 평균 97.4%이지만 실험값인 두 파라미터의 값이 커질수록 정확성은 떨어진다. 즉, CCR의 정확성은 실험이 진행될수록 낮아진다.

그림 9는 CCR의 실험 1에서 실험 4까지 각 응용 프로그램의 탐지성과 정확성의 변화를 나타낸 것이다. 응용 프로그램 eclipse-jdtcore의 탐지성은 모든 실험에서 동일하지만 이외의 모든 응용 프로그램에서는 파라미터의 값이 커질수록 탐지성은 증가하고 정확성은 감소한다. 그러나 각 응용 프로그램의 탐지성은 대체적으로 매우 완만하게 증가하는 반면 정확성은 매우 급하게 감소한다. 즉, 파라미터의 값을 도구의 기본값보다 큰 수로 변경하면 Bellon의 코드클론 표본 집합체의 쌍을 더 많이 찾기 때문에 CCR의 탐지성은 높아지지만 정확성은 탐지성의 증가율보다 더 급격하게 떨어진다.

따라서 도구의 탐지성은 일반적으로 높을수록 좋지만, 탐지성을 높이기 위해 파라미터의 값을

변경하면 오탐의 비율도 증가하기 때문에 도구 사용자는 이를 고려하여 적절할 실험값을 선택해야 한다. 클론 탐지 도구인 CCR에서 적절한 실험값은 실험 1이다. 이 실험의 파라미터 값은 다른 도구의 실험값과 거의 유사한 환경이므로 다른 도구와의 성능 비교에서 공평하다. 실험 1에서 CCR의 탐지성은 85.3%이며 정확성은 97.4%이다.

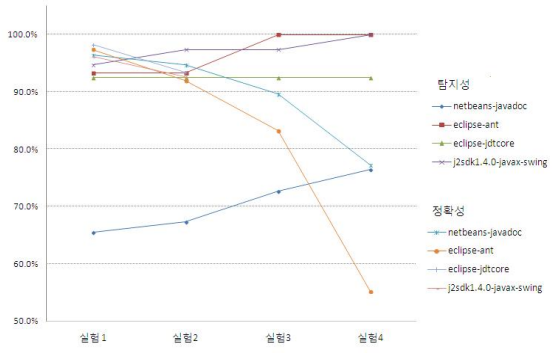


그림 9. CCR의 탐지성과 정확성 비교

5. 다른 도구와 탐지성 비교

CCR의 탐지성이 우수한지 알아보기 위해 Bellon의 코드클론 표본 집합체와 비교하여 탐지성을 계산한 기존의 코드클론 탐지 도구인 Asta[5]와 CloneDigger[3]의 결과를 비교한다. 두 도구는 CCR이 사용한 anti-unification 알고리즘을 기반으로 하지만 구현방법과 클론의 단위는 서로 다르다.

5.1 Evans의 Asta

Asta는 Evans-Fraser-Ma가 프로그램의 코드클론을 찾기 위해 anti-unification 알고리즘을 기반으로 소스코드의 클론 패턴을 탐지하는 도구이다[5]. Asta는 동적 계획법으로 구현하기 때문에 가능한 작은 크기의 클론을 찾아낸 뒤 이 클론들을 모아 큰 클론을 찾아내는 상향식 방법으로 클론을 찾는다. 이 도구는 Java와 C#으로 구현하였고 동일한 언어로 작성된 소스코드의 클론을 탐지한다.

Asta는 탐지성 확인을 위해 Bellon의 코드클론 표본 집합체와 비교하여 두 가지 항목인 ok-value와 good-value를 분석하였다. Ok-value는 기준인 클론 쌍이 다른 클론 쌍을 포함하는 비율이며 good-value는 두 클론 쌍의 전체 범위에서 공통 범위에 해당하는 비율이다. Asta는 두 값 ok-value와 good-value의 임계치를 0.7로 정했기 때문에 CCR도 Asta와 비교하기 위해 동일한 임계치인 0.7을 사용한다.

그림 10은 CCR과 Asta의 ok-value와 good-value의 비율을 그래프로 나타낸 것이다. CCR의 ok-value와 good-value는 앞의 4.3에서 계산한 CCR의 탐지성 중 실험 1의 결과이며 특히 CCR의 good-value는 탐지성과 동일한 의미이다. 그리고 Asta는 Evans-Fraser-Ma의 연구결과[5]이다. 그래프를 비교한 결과, CCR의 ok-value와 good-value는 Asta의 ok-value와 good-value보다 월등히 높다

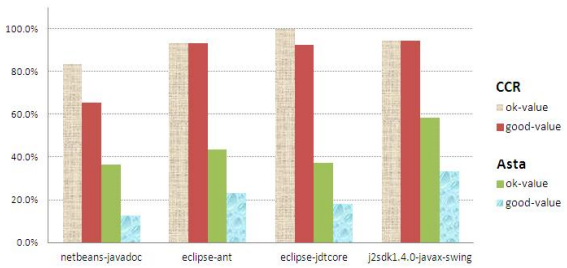


그림 10. CCR과 Asta의 ok-value와 good-value 비율

표 6은 비교 대상 집합체인 Bellon의 코드클론 표본 집합체와 CCR과 Asta의 ok-value와 good-value의 개수와 비율을 자세히 정리한 것이다. 비교결과 CCR은 Asta보다 응용 프로그램 netbeans-javadoc에서 ok-value가 약 2.3배, good-value가 5배 많고, 응용 프로그램 eclipse-

표 6. CCR과 Asta의 ok-value와 good-value

응용 프로그램	Bellon의 코드클론 표본 집합체	CCR		Asta	
		ok-value	good-value	ok-value	good-value
netbeans-javadoc	55	46 (83.6%)	36 (65.5%)	20 (36.7%)	7 (12.7%)
eclipse-ant	30	28 (93.3%)	28 (93.3%)	13 (43.4%)	7 (23.3%)
eclipse-jdtcore	67	67 (100.0%)	62 (92,5%)	- (37.4)%)	- (18.1%)
j2sdk1.4.0-javax-swing	38	36 (94.7%)	37 (97.4%)	- (58.4%)	- (33.2%)

ant에서는 ok-value가 약 2.2배, good-value가 4배 많다. CCR은 Bellon의 코드클론 표본 집합체의 5%를 대상으로 실험하였기 때문에 전체를 비교한 Asta와 단순한 수치 비교가 어려워 일부 응용 프로그램 eclipse-jdtcore와 j2sdk1.4.0-javax-swing에서는 두 값의 비율만 표시하였다. 비율로 확인한 CCR과 Asta의 good-value는 약 3.0-5.1배 그리고 ok-value는 약 1.6-2.3배 차이가 있다.

5.2 Bulychev의 CloneDigger

CloneDigger는 Bulychev-Minea이 anti-unification 알고리즘을 기반으로 개발한 코드클론 탐지 도구이다[3]. 이 도구는 파이썬으로 구현하였고 자바(java 1.5), 루아(Lua) 그리고 파이썬으로 작성된 소스코드의 클론을 탐지한다.

CloneDigger는 Bellon의 코드클론 표본 집합체와 비교하여 탐지성을 확인한다. 이때 네 개의 응용 프로그램에서 코드클론 쌍의 개수가 많아 비교가 어려운 응용 프로그램 eclipse-jdtcore와 j2sdk1.4.0-javax-swing은 클론의 5%만을 임의 선정하고 일부 클론은 삭제하여 비교 대상을 구축한다. 그러나 CCR은 원본인 Bellon의 집합체의 클론 유형과 동일한 비율을 유지하면서 임의로 제거하지 않고 전체 클론에서 일련번호가 큰 5%만을 비교 대상으로 선정하였다.

그림 11은 CCR과 CloneDigger의 탐지성을 비교한 그래프이다. CCR의 탐지성은 4.3의 실험 1의 결과이며 CloneDigger의 탐지성은 Bulychev-Minea의 연구[3]이다. 비교결과 모든 응용 프로그램에서 CCR의 탐지성은 CloneDigger보다 높다. 탐지성의 차이는 응용 프로그램 eclipse-ant에서 가장 적으며 eclipse-jdtcore에서 가장 크다.

그림 11. CCR과 CloneDigger의 탐지성

표 7은 CCR과 CloneDigger의 비교 대상인 표본 집합체와 탐지성을 자세히 나타낸 것이다. CCR의 탐지성과 CloneDigger의 탐지성을 비교 정리하면, CloneDigger의 비교 대상은 CCR의 비

표 7. CCR과 CloneDigger의 비교대상과 탐지성

응용 프로그램	CCR		CloneDigger	
	표본 집합체	탐지성	표본 집합체	탐지성
netbeans-javadoc	55	65.5%	46	50.0%
eclipse-ant	30	93.3%	22	88.3%
eclipse-jdtcore	67	92.5%	60	31.7%
j2sdk1.4.0-javax-swing	38	94.7%	39	48.7%
전체	190	85.3%	167	46.7%

교 대상보다 개수가 23개 적으면서 평균 탐지성도 1.8배 낮다. CCR의 탐지성이 CloneDigger의 탐지성보다 높은 이유는 동일한 알고리즘을 기반으로 하지만 클론의 단위가 문장이나 문장의 나열로 제한이 있기 때문이다. 그러나 CCR은 클론의 단위가 제한이 없기 때문에 다양한 패턴과 모양을 가진 클론을 찾는다.

6. 효율성

모든 실험은 Mac OX X Server 10.5.8, Xeon 2*2.8 Quad Core processor, 4GB RAM인 환경에서 실행하였다. 표 8은 각 응용 프로그램에서 수행한 CCR의 실행시간이다. CCR의 실행시간은 대상 프로그램의 크기가 커지고 파일의 개수가 많아지면 비교 횟수가 많아지기 때문에 조금 느려진다. 대략 3만 줄의 소스코드에서 코드클론을 찾는데 약 4분이 소요되고 20만 줄의 소스코드는 약 4시간 걸린다. 그리고 실험 4의 실행시간은 실험 1과 비교하여 1.2 - 1.5배 느리다.

도구의 실행시간은 실험환경에 영향을 받아 객관적인 판단이 어렵다. 따라서 기존의 도구 실행시간 외에 CCR의 성능을 복잡도로 표현하였다. CCR의 알고리즘은 크게 두 단계로 나눈다. CCR의 복잡도는 알고리즘 각 단계의 비교횟수를 계산한 뒤 더해서 구한다.

먼저 첫 번째 알고리즘인 패턴 모음에서는 프로그램 트리에서 가능한 모든 클론 패턴을 찾는다. 가능한 모든 트리 패턴을 찾기 위해서 두 트리의 모든 부분트리의 쌍을 찾아서 비교해야 한다. 비교해야할 부분트리 쌍의 개수는 두 트리의 노드 개수를 각각 m과 n이라고 할 때 $m \times n$이다. 그리고 하나의 트리 쌍으로부터 하나의 트리 조각을 찾는데 비교해야 할 노드의 개수는 똑같은 프로그램 트리를 비교하는 최악의 경우 $\min(m, n)$이다. 따라서 모양이 똑같은 프로그램을 비교하는 최악의 경우 비교횟수가 $O(m \times n \times \min(m, n))$이고, 완전히 다른 프로그램을 비교하는 최선의 경우 $O(m \times n)$이다.

두 번째 알고리즘인 패턴 클러스터링에서는 동일한 모양을 가진 클론패턴을 병합해서 클러스터 패턴을 생성한다. 클러스터 패턴은 탐지한 클론 패턴을 모아 놓은 패턴 모음에서 하나의 패턴을 뽑아 나머지와 비교하여 동일하면 병합하여 구축한다. 따라서 탐지한 클론 패턴을 모아 놓은 패턴 모음의 패턴 개수를 s라고 하면, 하나를 뽑아 나머지와 모두 비교할 횟수는 $\frac{s(s-1)}{2}$이다. 처음 선택한 패턴의 노드 개수가 p이고 두 번째 선택한 또 다른 패턴의 노드 개수가 q이면 동일한 패턴을 비교하는 최악의 경우 $\min(p, q)$이지만 일반적인 경우 이보다 훨씬 작다. 따라서 패턴을 병합하는 이 알고리즘은 최악의 경우 비교

표 8. 응용 프로그램에서 수행한 CCR의 실행시간

실험 번호		실험 1	실험 2	실험 3	실험 4	응용 프로그램의 줄 수
파라미터	MaxHole	15	15	25	25	
	HoleMassLimit	5	10	10	25	
netbeans-javadoc		0h 01m 47s	0h 02m 03s	0h 02m 17s	0h 02m 43s	14,301
eclipse-ant		0h 04m 16s	0h 03m 38s	0h 05m 05s	0h 05m 59s	29,880
eclipse-jdtcore		2h 42m 11s	2h 40m 47s	3h 14m 30s	3h 47m 36s	135,675
j2sdk1.4.0-javax-swing		3h 57m 30s	4h 00m 11s	4h 01m 05s	4h 12m 23s	202,943

횟수가 $O(\frac{s(s-1)}{2}\times\min(p,q))$이고, 최선의 경우 $O(\frac{s(s-1)}{2})$이다.

CCR의 두 단계 알고리즘에서 복잡도를 최악의 경우와 최선의 비교횟수로 정리하면 다음과 같다.

- 최악의 경우 비교횟수

  $O(m\times n\times\min(m,n)+\frac{s(s-1)}{2}\times\min(p,q))$

- 최선의 경우 비교횟수

  $O(m\times n+\frac{s(s-1)}{2})$

7. 결론 및 향후 연구방향

본 논문에서는 트리패턴 기반의 코드클론 탐지기인 CCR을 구현하였고 Java 응용 프로그램을 대상으로 실험하여 CCR의 정확성과 탐지성을 평가하였다. CCR은 파라미터의 값 변경으로 대상 응용 프로그램의 클론 유형 type 1과 type 2인 코드클론들을 모두 찾아내고 정확성과 탐지성이 높다. 특히 탐지성은 기존의 클론탐지 도구인 Asta보다 최대 5배 높고 CloneDigger보다 1.9배 더 높다.

향후 연구에서는 간격이 있는 클론(gapped clone 또는 non-contiguous clone)을 탐지하는 알고리즘을 고안하고 CCR에 적용할 예정이다. Bellon의 코드클론 표본 집합체에는 CCR의 파라미터 값을 변경해도 탐지하는 못하는 클론이 18개의 있는데, 이 클론이 모두 간격이 있는 클론이다. 예를 들어 프로그램 중간에 새로운 줄이 삽입되거나 삭제되어 여러 개의 조각으로 나뉜 패턴인지만 Bellon의 코드클론 표본 집합체는 하나의 클론으로 처리한다. 따라서 CCR의 탐지성을 더 높이기 위해서 간격이 있는 클론을 찾는 알고리즘을 개발하고 기존 도구에 추가해서 성능을 확인할 할 예정이다. 그리고 실행속도 향상을 위해 병렬처리의 적용도 고려해야 할 부분이다.

트리패턴을 기반으로 개발한 CCR은 정확성과 탐지성이 높기 때문에 프로그램의 유사성 판정이나 소프트웨어 감정 평가 및 리팩토링, 형상관리 등 적용범위가 넓을 것으로 기대한다.

참 고 문 헌

[1] Baxter, I., Yahin, A., Moura, L., and Anna, M., “Clone Detection Using Abstract Syntax Trees”, In ICSM, pp. 368-377, 1998.

[2] Bellon, S., Koschke, R., Krinke,J., and Merlo, E. “Comparison and Evaluation of Clone Detection Tools”, IEEE TSE, Vol.33(9), pp. 577-591, 2007.

[3] Bulychev, P. E. and Minea, M,, “An evaluation of duplicate code detection using anti-unification”, In IWSC, 2009.

[4] Burd, E. and Bailey, J., “Evaluating Clone Detection Tools for Use during Preventative Maintenance”, In ICSM, pp.35-43, Montreal, Canada, October 2002.

[5] Evans, W., Fraser, C., and Ma, F., “Clone Detection via Structural Abstraction”, In WCRE, pp.150-159, 2007.

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

[7] Kapser, C. and Godfrey, M., “Supporting the Analysis of Clones in Software Systems: A Case Study”, Journal of Software Maintenance and Evolution: Research and Practice, Vol. 18(2): 61-82, March 2006.

[8] Lee, Hyo-Sub and Doh, Kyoug-Goo,

"Tree-pattern-based duplicate code detection", In DSMM, pp.7-12, 2009.
[9] Li, H. and Tohmpson. S., "Incremental Code Detection and Elimination for Erlang Programs", In FASE, pp.356-370, 2011.
[10] Roy, C. K. and Cordy, J. R., "A survey on software clone detection research", Queen's School of computing TR 2007-541, 115 pp, 2007.
[11] Roy, C. K., Cordy, J. R., and Koschke, R., "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach", Journal of Software Maintenance and Evolution, Vol.21, iss. 2, pp. 143-169, Mar/Apr. 2009.

저 자 소 개

이 효 섭

1997년 대진대학교 컴퓨터공학과 학사
2000년 광운대학교 정보통신공학과 석사
2000년~현재 : 한양대학교 컴퓨터공학과 박사과정

<주관심분야 : 프로그래밍언어, 프로그램 분석 및 검증, 소프트웨어 보안>

도 경 구

1980년 한양대학교 산업공학과(학사)
1987년 미국 아이오와주립대학교 전산학과(석사)
1992년 미국 캔사스주립대학교 전산학과(박사)
1993.4~1995.8, 일본 Aizu 대학 교수
2000.9~2001.12, 스마트카드연구소 대표이사
2005.3~2006.2, 미국 캘리포니아대학 데이비스캠퍼스 방문교수
1995.9~현재, 한양대학교 안산캠퍼스 컴퓨터공학과 교수

<주관심 분야> 프로그래밍언어, 프로그램 분석, SW 보안, 프로그래밍언어 의미표기법