프로그램 유사성 검사기

장성순 서선애 이광근*
한국과학기술원 전산학전공

CloneChecker: A Program Similarity Checker

Sungsoon Jang, Sunae Seo and Kwangkeun Yi
Dept. of Computer Science, KAIST

요 약

표절을 쉽게 알아내기 위해, 프로그램 유사성 검사기(CloneChecker)를 만들었다. CloneChecker는 프로그램을 요약해서, 유사성을 계산하고, 비슷한 그룹들로 묶어 낸다. CloneChecker는 두 프로그램의 모든 부 구문트리(abstract syntax sub-tree)들을 서로 비교하므로 구문의 사소한 변화에 민감하지 않으며, 그럼에도 해쉬 함수를 이용하여 빠르게 수행된다. CloneChecker는 실제 강의에서 사용되었으며, C, Java, Scheme, nML로 짜여진 프로그램들에 대해 동작한다.

1 서론

프로그램의 표절은 공공연히 일어나지만 찾기가 쉽지 않다. 크게 회사에서 쓰이는 상용 프로그램의 도용에서부터, 작게는 학생들이 내는 프로그램 숙제에 이르기까지 여러 분야에서 표절이 일어난다. 표절을 찾으려면, 일일이 눈으로 확인해 보는 것이 제일 좋은 방법이나, 프로그램이 커지고, 많아지게 되면서 이는 매우 어려운 일이 되고 있다.

표절을 쉽게 알아내기 위해서 프로그램 유사성 검사기(CloneChecker)를 만들었다. 유사성이란 대략 비교하는 두 프로그램의 구문트리(abstract syntax tree) 전체에서 공통된 구문 트리들이 차지하는 비율이다. CloneChecker는 프로그램들을 비교하여 비슷한 프로그램끼리 그룹으로 묶어낸다. CloneChecker는 강의 중에 프로그램 숙제의 표절을 막기 위해 처음 고안되었다. 표절은 그대로 복사하는 것에서 부터, 변수 이름 바꾸기, 함수 순서 바꾸기등 표절을 쉽게 찾아내지 못하도록 고의적으로 변화를 준다. 이러한 고의적 변화에 유사성 값은 민감하지 않도록 만들었다.

CloneChecker는 nML[1]로 짜여졌다. nML은 ML의 한 종류로서 현재 프로그램 분석 시스템 연구단에서 개발 중이다. ML자체가 언어 분석기를 짜는데 강력하기 때문에, CloneChecker 역시 작고, 쉽게 짤 수 있으면서도, 복잡한 프로그램의 분석을 수행해낸다.

2 방법

CloneChecker는 크게 요약, 유사성 비교, 그룹짓기의 세 과정을 거친다.

2.1 요약

프로그램을 비교하기 전에 요약한다. 요약은 각각의 프로그램 언어마다 조금씩 다른데 대체로 다음과 같다.

- 프로그램 언어는 몇 개의 문법적 범주(syntactic category)를 가지고 있는데, 이를 한 개의 문법적 범주를 가진 나무구조(tree)로 바꾼다.
- 모든 식별자(identifier)를 동일한 식별자로 본다.
- 모든 참, 거짓(boolean) 값을 동일하게 본다.
- 모든 정수(integer)는 동일한 정수로 본다.
- 모든 실수(float)를 동일한 실수로 본다.
- 타입(type) 정보는 무시한다.
- 프로그램 식(expression)은 대부분 그대로 옮기지만, 언어에 따라 어떤 것은 무시하거나, 생략한다.

위에서 바뀌는 나무구조의 타입(type)은 아래와 같다.

```
type exp = LEAF of int
         | NODE of int * exp list
         | VACANT_NODE of int * exp list
```

위 타입에서 구성자 옆의 정수는 각 잎새나 노드의 종류이다. 예를 들어 모든 식별자는 LEAF 1 참,거짓 값은 LEAF 2등으로 바뀌며, if 프로그램 식은 NODE(7, sub_trees)로, while 프로그램 식은 NODE(8, sub_trees)로 바뀐다. 여기서, VACANT_NODE는 아래에서 유사성을 계산할 때 더해지지 않는 노드이다. 하지만, 이 노드의 부분 트리(sub-tree)들은 고려되며 이 노드는 단지 부분 트리들을 묶어주는 역할을 한다. 이는 C의 선언식(declaration)등에 사용되며, 선언되는 함수들의 순서를 바꿔도 유사성이 같게 나오도록 해준다.

프로그램을 요약하면 다음의 장점이 있다.

- CloneChecker가 작고, 간단하고, 빨라진다. 문법적 범주가 여러개라면, 부분 트리끼리 비교를 수행하는 CloneChecker는 커지고 복잡해지며, 대략 문법적 범주 갯수의 제곱에 비례해서 프로그램이 커진다. 잎새가 요약되어서 잎새의 비교 시간이 짧아지는데, 나무구조

*본 연구는 과학기술부 창의적연구진흥사업 추진으로 얻어진 결과임.
†E-mail: {abyss,saseo, kwang}@ropas.kaist.ac.kr

$$Similarity(a,b) = \frac{S_a + S_b}{T_a + T_b}$$

S_a = number of sub-trees of a that appears identically in b
S_b = number of sub-trees of b that appears identically in a
T_a = number of total nodes(sub-trees) in tree a
T_b = number of total nodes(sub-trees) in tree b

그림 1: 유사성 정의

에서 잎새가 차지하는 비율이 크기 때문에 이는 전체적으로 크게 속도를 향상시킨다. 즉, 비교속도가 빨라진다.

- 원본을 약간만 수정해서 표절한 프로그램의 경우, CloneChecker는 원본과 복사본이 완전히 같다고 알려줄 수 있다. 변수나 함수 이름만 바꿔칠 경우, 식별자는 모두 같은 잎새로 변환되었으므로 영향이 없다. 함수 선언 순서를 바꿨을 경우 위의 VACANT_NODE를 사용해서 유사성 비교할때 영향을 주지 않는다. 이는 2.2 절에서 자세히 설명하도록 하겠다.

프로그램의 요약은 안전하지 않다. 위의 프로그램 요약은 많은 정보를 생략해서 서로 다른 프로그램이라도 CloneChecker가 같다고 말할 수 있다. 그러나, CloneChecker의 목적은 위에서 얘기한 그대로 복사한 것에서 미미하게만 나아간 수준의 표절을 완벽하게 잡아내는 것이 아니라, 길고도 많은 수의 프로그램에서 표절로 생각되는 그룹들을 추려내주고, 상당히 지능적인 표절에서도 유사성을 높게 나오도록 하는데 있다.

2.2 유사성

유사성(similarity)은 그림 1에 정의되어있다. 유사성은 두 프로그램이 비슷한 정도를 나타내는 값인데, 0과 1사이의 실수이다. 여기서, 숫자가 커질 수록 두 프로그램은 비슷하며, 유사성이 1.0이란 것은 두 프로그램의 요약이 완전히 같다는 것을 나타낸다. 정의를 간단히 설명하자면, 두 프로그램에서 반대편의 프로그램에서 동일한 부분 트리(sub-tree)가 존재하는 모든 부분 트리들이 두 프로그램의 전체 노드 수(전체 부분 트리의 수)에서 차지하는 비율이다. 그림 1에서 S_a와 S_b는 다를 수도 있다.

유사성을 계산하는 것은 큰 시간 복잡도(time complexity)를 가진다. 여기서, 두 노드가 같다는 것은 두 노드를 시작점으로 하는 두개의 부분 트리가 같은 것을 의미한다. 모든 부분 트리를 서로 비교하는 단순한 알고리즘을 생각한다면, 트리의 노드수가 N일때 두 부분 트리의 비교가 복잡도 $O(N)$이므로 서로 N^2번 반복해서 시간 복잡도가 $O(N^3)$이 된다. 경험적으로 M줄의 프로그램은 약 $10 \times M$개의 노드들을 갖는다. 따라서, 100줄의 프로그램의 경우 N^3이 10^9이 된다.

Baxter[2]들이 제시한 것 처럼, 부분 트리들을 해시(hash) 값에 따라 나누어 놓으면 비교하는 시간을 크게 줄일 수 있다. 즉, 해시값이 다른 노드들은 같은 노드일 수 없으므로, 비교할 대상이 줄어든다. 해시값이 잘 정의되어 있다면, 비교

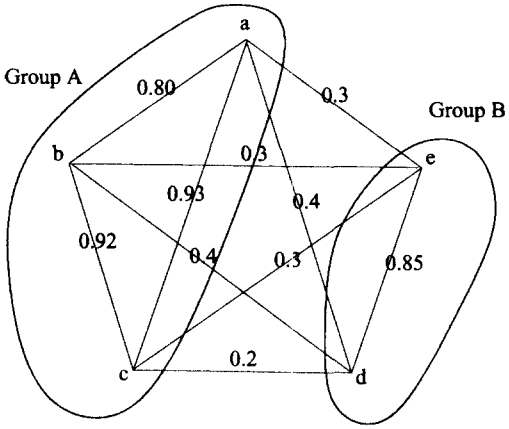

그림 2: 전역 유사성이 8.0일때 그룹짓기의 예: 여기서 a, b, c, d, e는 프로그램이고, 프로그램을 잇는 선 위의 숫자는 유사성이다.

를 N번만 반복하면 되므로, $O(N^2)$의 시간 복잡도를 가진다. 초기 CloneChecker에서 100 개의 100 줄의 nML 프로그램이 있을때, 서로간의 비교는 SUN Sparc Ultra-4에서 두시간이 걸렸었다. 최근의 버전은 앞에서와 동일한 조건에서 일분 안에 끝났다.

앞의 VACANT_NODE로 시작하는 트리는 부분 트리의 순서를 바꿔도 완벽하게 같다고 나온다. 앞에서 말했듯이 VACANT_NODE들은 유사성 계산에서 제외된다. 즉, S_a, S_b, T_a, T_b중 어떤 수에도 들어가지 않고, 단지 아래의 부분 트리들을 묶어주는 역할을 한다. VACANT_NODE로 시작하는 트리 A가 부분 트리 a, b를 가지고 있자고 하자. 이제 a, b의 순서를 바꾼 트리 A'은 A와 다른 트리가 된다. 하지만, 그림 1에서 처럼 계산하면 분자는 $2 \times (a$의 크기 $+ b$의 크기$)$가 되고, 분모 역시 A와 A'의 시작 노드가 들어가지 않으므로, $2 \times (a$의 크기 $+ b$의 크기$)$가 되어서 유사성이 1.0이 된다.

2.3 그룹짓기

비교되는 프로그램들의 모든 쌍에 대해서 각각 유사성을 계산한다. 프로그램이 N개라면 모두 $N(N+1)/2$ 번의 유사성 계산을 수행하는 것인데, 이는 위에서 얘기한 알고리즘을 사용하면 그다지 시간이 많이 들지 않는다.

프로그램은 기본적으로 다음과 같이 그룹짓는다. 그룹안의 임의의 두 프로그램의 유사성은 0과 1사이의 실수인 전역 유사성(Global Similarity)보다 큰 값을 갖게 된다.

Definition 1 *전역 유사성 g에 의해 정의된 그룹 G는*

$$\forall a \in G. \forall b \in G.\ Similarity(a,b) \geq g$$

이다.

그러나, 전역 유사성으로 된 그룹짓는 방법은 같은 프로그램이라도 경우에 따라 달리 그룹지어질 수도 있다. 또한, 비록 위의 정의를 만족시키지 않아도, 어떤 프로그램은 그룹의 거의 모든 프로그램과 비슷할 수 있다. 이 때, 이 프로그램을 그룹에 넣을지 넣지 않을지는 애매한 문제이다. 그래서, 아래에

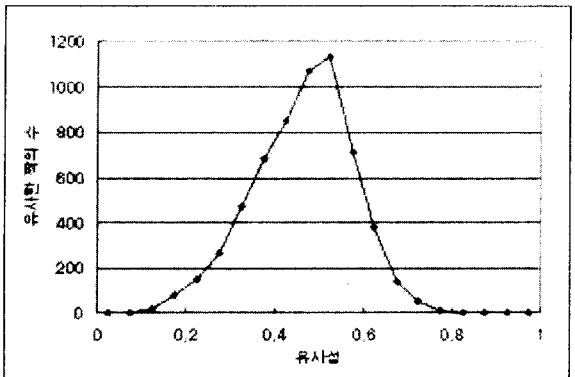

그림 3: 특정한 문제(vocalize 함수짜기)에 대하여 110명의 학생들이 제출한 숙제들 사이의 유사성 히스토그램

정의된 지역 유사성(Local Similarity)을 기준으로 이를 보완한다.

Definition 2 *지역 유사성 l에 의해 정의된 그룹 G는*

$$\forall a \in G. \exists b \in G.\ Similarity(a,b) \geq l$$

이다.

3 파서

CloneChecker는 C, Java, Scheme, nML의 네개의 언어를 지원한다. CloneChecker는 다양한 언어로 된 프로그램들을 파싱(parsing)하는 부분과 이들을 비교하는 부분으로 나누어져 있다. 각각의 파서는 nML로 짜여졌으며, 각각 따르는 표준은 아래와 같다.

- C언어 파서는 Standard ML(SML)로 짜여진 C언어 파서인 ckit을 nML에 맞게 수정하였다.
- Java 파서는 CUP이라고 불리는 Java로 짜여진 Java 파서를 참조했다.
- Scheme은 표준 방법(Revised(5) Report on the Algorihtmic Language Scheme [3])을 따라서 구현했다.
- nML은 nML 컴파일러[1]에서 사용하는 파서를 그대로 사용하였다.

4 결과

지난 2001년도 봄 학기 프로그램 언어 강의[4]에서 학생들의 nML 프로그램들에 대해서 유사성 검사를 수행하였다. 121명의 학생들에게 모두 일곱번의 프로그램 숙제를 제출했다

그 유사성 검사 결과 CloneChecker는 표절을 찾아내는 일을 비교적 잘 수행했다.

- 유사성이 높을 수록 프로그램은 비슷했다. 특히 유사성이 1.0인 경우엔 변수 이름이 바뀌었거나 함수 선언 순서가 바뀐 것 이외에는 예외없이 동일한 프로그램이었다. 프로그램의 크기가 작거나, 수업시간에 배운 알고리즘을 사용해서 프로그램의 형태가 비슷해질 수록 유사성이 전체적으로 높게 나왔다.
- 프로그램들의 유사성 분포가 그림 3과 같다. 즉, 모든 프로그램들은 어느 이상의 유사성을 가지고 있었으며, 가우시안(Gaussian) 분포를 가졌다. 아주 유사성이 높은 프로그램이 나올 확률이 대개 매우 낮다고 말할 수 있다. 그러므로, 그림 3에서 유사성이 매우 높은 프로그램들은 충분히 표절이라고 의심해 볼 수 있다. 그림 3에서 사용된 문제는 vocalize라는 함수를 짜는 것으로서, 이 함수는 일곱자리 양의 정수 string을 받아서 세자리와 네자리로 나누어서 읽어주도록 하는 함수이다.

학생들의 의견은 다음과 같았다. 몇몇 학생은 프로그램을 스스로의 힘으로만 했는데도 CloneChecker에 걸렸다고 불평했다. 이 경우는 프로그램이 간단하거나 짜는 형식이 거의 일정하거나, 아니면 우연히 같은 경우였다. 몇몇 학생은 실제로 카피를 했고, 이것이 CloneChecker에 적발되어 카피를 시인하였다. 심리적으로 재미있는 일은, CloneChecker를 사용한 후부터는 학생들이 맘 놓고 서로 프로그램 숙제를 주는 일이 줄어들었다는 것이다.

CloneChecker는 요약해서 비교하기 때문에 완전히 정확하지는 않으나, 표절의 가늠자가 될 수 있고, 실제 적용에서 긍정적 결과를 주었다. CloneChecker는 함수형 언어인 ML로 짜여져서 작고, 간단하면서도 강력하다. CloneChecker는 KAIST LET 프로젝트의 일부이며, `http://ropas.kaist.ac.kr/n/clonechecker`에서 다운 받을 수 있다.

참고 자료

[1] nml programming language system, 2001. `http://ropas.kaist.ac.kr/n/`.

[2] Ira D.Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant'Anna, and Lorraine Bier. Clone detection using abstract syntax tree. In *Proceedings of the ICSM'98*, November 1998.

[3] H. Abelson et. al. Revised report on the algorithmic language scheme. *LISP and Symbolic Computation*, 11:p7–105, August 1998.

[4] 이광근. Cs320 programming languages, 2001. `http://ropas.kaist.ac.kr/~kwang/320/01`.