프로그램 표절 검출 방법에 대한 조사

이효섭° 도경구
한양대학교 컴퓨터공학과
{leehs°, doh}@cse.hanyang.ac.kr

Survey on Program Clone Detection Methods

Hyo-Sub Lee° Kyung-Goo Doh
Department of Computer Science & Engineering, Hanyang University

요 약

표절이란 원본 자료의 출처를 밝히지 않고 사용하는 행위인데, 프로그램 표절은 표절로 의심되는 유사 소스코드를 말한다. 프로그램 표절을 검출하는 방법은 소스코드의 토큰 요약 후 비교하거나 단어와 키워드의 개수를 측정해 통계적으로 비교, 계산하거나 또는 프로그램의 구문구조를 비교하는 등 여러 접근 방법이 있다. 본 조사 보고서에서는 프로그램 표절 검출 방법 및 응용 도구에 대한 최근 연구 동향을 조사하여 정리하고 향후 발전 방향을 토론한다.

1. 서 론

표절은 원본자료의 출처를 분명히 밝히지 않고 자신의 것처럼 사용하는 행위[1]를 말하는데, 학생이 제출하는 과제에서부터 산업 현장의 프로그램 일부 또는 전체에 이르기까지[2] 프로그램 표절 범위는 매우 다양하다. 표절로 의심되는 프로그램 소스코드는 자동화 된 표절 검출도구를 이용해 평가하지만, 어느 정도 유사해야 표절로 판정하는가가 어렵기 때문에 빠른 코드 비교와 동시에 결과의 신뢰성도 높일 수 있는 표절 검출 방법이 필요하다.

본 논문에서는 기존 연구된 프로그램 표절 공격 및 검출 방법과 최근 연구 동향에 대해 알아보고 향후 연구방향에 대해 논의한다.

2. 표절 공격 방법

프로그램에서 변수의 이름을 바꾸거나 값을 수정하는 간단한 표절 공격은 쉬우나, 소스코드에 대한 철저한 이해가 필요한 전체적인 구조 변경은 어렵다. 프로그램의 의미를 이해하지 못한 상태에서의 소스코드 구조변경은 예상치 못한 오류를 발생하거나 또는 프로그램이 원하는 결과물을 얻지 못할 가능성을 높이기 때문이다.

1) 원시 소스 그대로 복사
2) 주석의 삽입, 삭제, 편집
3) 함수나 변수의 이름, 타입, 값 변경
4) 불필요한 코드의 삽입
5) 선언된 함수나 변수의 순서 변경
6) 부분적인 코드 변경
7) 조건문의 구조 변경
8) 다른 코드들의 조합
9) 특정 함수 또는 모듈을 분할하여 코드 생성
10) 여러 함수 또는 모듈을 통합하여 코드 생성

표 1 프로그램의 표절 방법

표 1에 Whale[3]이 제시한 일반적인 프로그램 표절 방법 10가지를 나열해 놓았다. 최근에는 이러한 표절 공격 방법도 점차 발달하여 주석이나 변수와 같은 단순한 변경에서 루틴 및 모듈, 라이브러리의 변환과 같은 복잡한 변경으로 확대되고 있다. 또한 의미가 전체적으로 정확히 일치하는 표절 공격뿐만 아니라, 의미는 정확하게 일치하지 않더라도 프로그램의 실행에 크게 영향을 미치지 않는 범위 내에서 코드를 변형하거나 부분적으로만 코드를 복제하는 공격도 늘고 있다.

해당 언어가 구체적일 경우 언어의 문법적 특징에 따른 특수한 표절 공격 방법이 존재한다. 코볼이나 포트란, 베이식 등의 언어는 고유한 코드 용지 사용법이 있어 해당 영역에 프로그래밍을 해야 하는데, 위치나 줄 번호 변경으로 간단한 표절 공격이 가능하다. 이와 같이 일반 표절 공격 방법과 함께 언어에 구체적인 특징을 이용한 표절 공격방법이 확장되어 사용된다.

3. 표절 검출 방법

프로그램 표절의 검출 방법에는 통계적인 수치를 이용하는 방법과 구조를 비교하는 방법이 있다.

통계 측정 방법은 두 프로그램에서 사용된 단어나 키워드의 유사성 및 사용 빈도를 비교하는 방법으로 문서의 길이에 영향을 받지 않으면서 특징을 찾기 용이하다는 장점이 있다. Plagiarism.org[4], Intergri-guard[5], SCAM, EVE2[6], Windiff, CopyCatch[7], Glatt Plagiarism Service Inc[8], COPS[9], SIM[10], Siff[11] 등이 이 방법을 이용하여 다양한 응용 프로그램을 만들었는데, 이론적 바탕은 Halstead의 소프트웨어 과학 측정법[12]과 McCabe의 싸이클로매틱 복잡도[13]에 두고 있다.

구조를 이용하는 방법은 프로그래밍 언어의 구문구조를 주로 이용한다. 구조분석으로 표절을 검출하는 방법은 크게 세 가지로 세분된다. 어휘를 분석해 토큰을 추출한 뒤 나열된 토큰을 요약하여 일차원적으로 비교하는 방법, 함수의 호출 순서를 분석하는 함수 호출 선형화

방법, 그리고 프로그램 복원(parsing)을 통해서 얻은 프로그램 트리 구조를 이차원적으로 비교하는 방법이 있다. 개발된 검출도구로는 토큰 비교 방식의 Plague[14], YAP[15], YAP3[16], MOSS[17], Jplag[18], Bandit[19], SABRE BinDiff[20], CodeMatch[21], WcopyFind[22], Cogger와 함수 호출 선형화의 ESPA[23] 그리고 프로그램 트리 구조 비교인 CHECK[24], Clone-Checker[25,26] 등이 있다.

표 1에서 제시한 방법을 사용한 프로그램 표절은 구조분석을 통해 일부 검출가능한데, 다양한 프로그램 표절 공격에 대하여 표절 여부를 검출할 수 있는 구조분석 방법을 짝지어보면 표 2와 같다.

표절 검출 방법	검출 가능한 표절 공격
토큰요약 비교	2) 주석의 삽입, 삭제 등 편집 3) 함수나 변수의 이름 및 타입, 값 변경
함수호출선형화 비교	4) 불필요한 코드의 삽입 5) 선언된 함수나 변수의 순서 변경 8) 다른 코드들의 조합 9) 특정 함수, 모듈을 분할하여 코드 생성 10) 여러 함수, 모듈의 통합으로 코드 생성
프로그램 트리구조 비교	6) 부분적인 코드 변경 (일부만 가능) 7) 조건문의 구조 변경

표 2 구조분석을 통한 프로그램 표절 검출 방법

토큰요약은 모든 참, 거짓 값을 동일하게 하며, 모든 정수를 동일하게 본다. 실수와 식별자도 모두 동일한 실수, 식별자로 인식하고 타입 정보도 무시하기 때문에 요약된 어휘분석은 주석문의 삽입, 변경, 삭제를 통한 표절이나 변수의 이름 및 타입, 값 변경 시도에 유연하게 대처할 수 있다. 함수 호출 선형화는 기본 알고리즘은 같으나 유사한 기능을 하는 함수를 호출하거나 또는 중요 알고리즘은 같으나 인터페이스 처리 부분만 변경한 경우에 검출이 용이하며 특히 하나의 함수나 모듈을 여러 개로 분할하거나 여러 개의 함수나 모듈을 하나로 통합한 경우 탐색이 가능하다. 프로그램의 의미에 영향을 주지 않는 불필요한 코드를 삽입하는 표절도 검출할 수 있다. 프로그램 트리 비교 방법은 선언된 함수들의 순서를 바꾸는 구조 변경에 따른 표절과 조건문의 구조 변경에 따른 표절 공격을 감지 할 수 있다.

4. 최근 연구 동향

최근 프로그램 표절 검출에 대한 연구는 특정 분야에 사용되거나 한 가지 프로그램 언어의 특성과 패턴에 적합한 검출방법을 상세히 고려하면서도 타 언어간 비교가 가능하도록 하는데 초점이 맞춰지고 있으며, 그에 맞는 자동화된 도구가 개발되고 있다.

여러 프로그래밍 언어가 하나의 서비스를 제공할 때 정확한 표절 검출이 어려울 수 있다. 인터넷 서비스를 제공하는 ASP의 경우 주로 HTML 문서 내부에 자바스크립트 또는 비주얼베이식 코드가 삽입된 형태로 이루어져 있으며, 각 스크립트 프로그램의 구문구조는 서로 다르다. 빈칸의 삭제 및 추가는 매우 손쉬운 표절 방법이지만, 여러 언어가 혼합된 서비스에서는 모든 빈칸의 변경이 언어에 따라 다른 의미를 내포할 수 있기 때문에 빈칸에 민감한 파서 선택이 필요하다. 또한 주석은 서로 다른 문법으로 처리되므로 문법적 비교보다는 패턴 매칭으로 검출하여 나은 성능평가를 한다[27].

표절의 단계 또한 상세하게 분류한다. 과거에도 두 파일의 유사도 비율에 따라 표절 여부를 판단했지만, Calefato[28]는 유사도 뿐 아니라 웹 응용 프로그램이 가지는 특징을 기준으로 동일(Identical), 거의동일(Nearly-identical), 유사(Similar), 다름(Distinct)과 같이 4가지 상태로 나누어 표절 정도를 표현한다. 분석 결과는 그래프로 나타내거나[29] 비교표를 만들기도 하는데, 4단계의 표절 가능성 분류와 표절 도구로 검출한 결과를 실제 존재하는 전체 표절 비율로 계산해 결과의 완전성(Recall)과 정확성(Precision)을 기준으로 분류하고 평가한다[30].

사용언어가 한가지일 경우 명확한 표절 정의 및 경향 분석은 정확한 표절 검출에 도움을 준다. Kamiya[31]는 객체지향 프로그래밍 언어인 Java와 C++의 표절 검사를 위해 언어에 적합한 패턴 매칭 방법을 룰(rule)로 구조화한 뒤 그래프로 표절 결과를 확인한다. Baxter[32]는 Java에서의 프로그램 트리구조 비교와 동시에 버그 검사를 함으로써 동일한 오류가 발생한 파일을 의심하는 역추적하는 방식을 이용해 표절을 검출한다.

대부분 표절 도구에서 지원되는 언어의 종류는 한정되어 있어 다른 언어로 동일한 기능을 작성하면 표절로 판단하기 어렵다. 자동화 도구인 CloneDoctor[33]는 프로그램 트리와 해시코드를 이용해 거의 모든 언어와 대용량 소스에 적용가능하며, Cordy[34]는 유사 가능성이 있는 목록을 추출 후 비교를 통해 특정한 언어에 독립적이면서 유사 표절도 검사한다. Cordy의 핵심기능인 TXL은 파싱을 통해 Island라는 언어에 종속된 구조를 만들고 일정 형식을 가진 형태로 작성해 프로그램 소스가 가지는 특징을 제거함으로써 더 나은 비교 결과를 찾는다.

5. 적용분야

표절 검사를 적용하는 분야는 점점 다양화되고 있다. 가장 활발하게 사용되고 있는 분야는 대학에서 학생들의 프로그램 과제검사이다. 과제의 공정한 평가를 위해 다양하고 특색 있는 알고리즘을 탑재한 표절 검출도구[6,7,8,10,12,13,19,25,29]가 개발되어 공개되었다. 기존의 도구가 나름대로 일정 수준의 판별력을 보여주고 있지만, 그 중 프로그램 트리구조를 이용한 방법은 특히 프로그램의 구조 정보를 추가적으로 이용해 교묘한 공격의 경우에도 판별력을 높일 수 있으며, 실제 Clone-Checker [26]는 특정 패턴에 대해서 판별력이 우수한 것으로 나타나고 있다.

두 번째는 개발자 코드의 재사용과 유지보수를 좋게 하기 위해 사용한다. 표절 검사를 통해 유사코드를 발견하면 삭제하거나 프로시저로 만들어 사용한다[33]. 숨은 중복코드를 찾아내는 것은 버그를 감소시키고 코드의 사이즈를 축소함으로써 코드의 복잡도를 줄이고 효율성을 높일 수 있다.

그 외 표절 검증도구는 회사에서의 소스 버전관리[19]

나 컨설팅회사에서의 기능검증[33]으로도 사용되고 있다.

6. 향후 연구 및 결론

지금까지 알려진 표절 기법과 검출도구를 알아보고 최근 연구 동향도 살펴보았다. 다양한 표절 검출도구는 표절 가능성을 감지해 수치적으로 표시하는 것 뿐 아니라 심리적인 부담감을 준다는 장점도 가진다. 그러나 문제점은 검출도구가 있다 할지라도 표절내용을 완벽하게 잡아내지 못한다는 것이며, 표절판정을 내리는 범위 설정이 어렵다는 것에 있다. 프로그램을 대상으로 했을 때 두 소스의 의미가 비슷하면 표절로 분류할 것인지 아니면 모양새가 비슷한 것만을 표절로 분류해야하는지와 같은 표절에 대한 명확한 정의나 한계가 분명하지 않다.

추후 연구되어야 할 부분은 표절에 대한 프로그래밍 언어의 문법구조와 의미 정의 그리고, 아직까지 표절 검출이 어려운 유사형태나 부분 코드 일치 공격에도 대응할 수 있는 분석방법과 검출도구가 있어야 한다. 특히 표절 검출 도구는 일반 표절 내역을 검사함과 동시에 타 언어간 비교와 분석이 가능하도록 확장 가능하며, 해당 언어에서만 발견되는 문법적 특징을 이용한 표절도 검출할 수 있어야 할 것이다. 그리고 보다 많은 사용을 위해 도구 개발에 있어 GUI측면도 고려해야 할 것이다.

7. 참고문헌

[1] J. R. Edlun, "What is "Plagiarism" and why do people do it?", University Writing Centre Director, California State University, LA, 1998

[2] 전명제, 이평준, 조환규, "학생 프로그램 과제물 표절 탐색 기법", 한국 S/W 감정평가학회 춘계학술대회, pp83-88, 2004

[3] G. Whale, "Identification of program Similarity in Large Populations", The Computer Journal, Vol. 33, Number 2, pp. 140-146,1990

[4] http://www.plagiarism.org

[5] http://www.integriguard.com

[6] http://www.canexus.com/eve/abouteve.html

[7] http://www.copycathgold.com

[8] http://www.plagiarism.com/

[9] S. Brin, J. Davis, and H. Garcia-Molina, "Copy detection mechanisms for digital documents", In proceedings of the ACM SIGMOD Annual Conference, CA, May 1995

[10] http://www.few.vu.nl/~dick/sim.html

[11] http://glimpse.arizona.edu/jaavadup.html

[12] M. A. Halstead, "Elements of Software Science", North Holland, New York, 1977

[13] T. J. McCabe, "A complexity measure", IEEE Transaction on Software Engineering, Vol. SE-2, No.4, pp.308-320, December 1976

[14] G. Whale, "Identification of Program Similarity in Large populations", The Computer Journal, Vol.33, Number 2, 1990

[15] M. J. Wise, "Detection of similarities in student programs : YAP'ing maybe preferable to Plague'ing", SIGSCI Technical Symposium, pp.268-274, March 5-6, 1992

[16] M. J. Wise, "YAP3 : improved detection of similarities in computer programs and other texts", In proceeding of SIGCSE'96, pp.13-134, 1996

[17] http://www.cs.berkeley.edu/~aiken/moss.html

[18] http://wwwipd.ira.yka.de:2222/

[19] http://www.goldleafdesigns.co.uk/~adrian/

[20] http://sabre-security.com/

[21] http://www.zeidmanconsulting.com/codematch.htm

[22] http://plagiarism.phys.qirqlinia.edu/

[23] http://pearl.cs.pusan.ac.kr/~prog2004/

[24] A. Si, H. V. Leong, and R. W. H. Lau, "CHECK : A document plagiarism detection system", In Proceedings of ACM Symposium for Applied Computing, pp.70-77, 1997

[25] 장성순, 서선애, 이광근, "프로그램 유사성 검사기", 정보과학회 가을 학술대회, 제28권 2호, pp. 334-336, 2001년 10월,

[26] http://ropas.snu.ac.kr/n/clonechecker/

[27] Nikita Synytskyy, James R. Cordy, Thoma R. Dean, "Robust Multilingual Parsing Using Island Grammars", CASCON 2003 : 266-278.

[28] F. Calefato, F. Lanubile, and T. Mallardo, "Function Clone Detection in Web Applications: A Semi-automated Approach", Journal of Web Engineering, vol.3, no.1, Rinton Press, May 2004, pp. 3-21

[29] K. Kontogiannis, "Evaluation Experiments on the Detection of Programming patterns Using Software Metrics", in Fourth Working Conference on Reverse Engineering, Amsterdam, The Netherlans, 1997, pp.44-54

[30] J. Krinke, "Identifying Similar Code with Program Dependence Graphs", In Eighth Working Conference on Reverse Engineering, Stuttgart, Germany, 2001, pp. 301-309

[31] T. Kamiya, S. Kusumoto, K. Inoue, "CCFinder : A multi-linguistic token based code clone detection system for large scale source code", IEEE, 2002

[32] I. Baxter, Ira D. & Churchett, Dale. "Using Clone Detecion to Manage a Product Line", ICSR 7 Workshop, April 16, 2002

[33] I. Baxter, C. Pidgeon, and M. Mehlich. DMS : Program Transformation for Practical Scalable Software Evolution. In Proc. Int. Conf. Software Engineering (ICSE), 2004, pp. 625-634.

[34] James R. Cordy, Thomas R. Dean, Nikita Synytskyy, "Practical Language-Independent Detection of Near-Miss Clones", Proceeding of the 2004 conference on Collaborative research, pp. 1-12, 2004