프로그램 부분 복제 검출 기법

김연어*, 이필수*, 우균*
*부산대학교 컴퓨터공학과
e-mail:{yeoneo, skyz1104, woogyun}@pusan.ac.kr

A Method for Detecting Partially Clone on the Programs

Yeoneo Kim*, Pil-Su Lee*, Gyun Woo*
*Department of CSE, Pusan National University

요 약

최근 대학가의 과제물 복제는 사회적인 이슈가 되고 있다. 그 중 프로그래밍 과제에 대한 복제는 디지털 복제물의 특성상 빈번히 발생하며 그만큼 많이 연구되고 있는 분야이다. 이 논문은 기존의 프로그램 복제 검출 기법이 해결하지 못하는 프로그램 부분 복제 문제를 해결하는 방법을 제시한다. 프로그램 부분 복제 문제란 여러 원본 프로그램으로부터 프로그램을 복제하는 문제를 의미한다. 이 논문에서는 실제 프로그램을 통해 부분 복제 문제를 보인다. 그리고 이 문제를 해결하는 방법으로 프로그램 간의 유사한 구간 정보를 이용하는 것을 제시하고 간단한 프로그램을 대상으로 제안 기법을 적용해 프로그램 부분 복제 문제를 검출하는 결과를 보인다.

1. 서론

최근 대학가의 과제물에 대한 복제 문제는 심각하게 다루어지고 있다[1,2]. 특히 과제물에 대한 복제 문제 중 프로그래밍 과제에 대한 복제는 손쉽게 이루어진다. 이는 프로그램 코드는 디지털 저작물의 특성을 가지며 디지털 저작물은 간단한 명령어 몇 개만으로 손쉽게 복제가 이루어진다. 그리고 이로 인해 프로그램 및 소스 코드의 복제는 빈번히 발생할 수 있다. 또한, 프로그램 복제 문제는 과제뿐만 아니라 상업적 프로그램에서도 발생한다. 상업적 프로그램의 문제는 주로 라이선스 문제에서 발생한다. 이 문제는 상업적 프로그램이 무단으로 오픈 소스 코드를 사용함으로 라이선스의 범위를 지키지 않기 때문에 발생한다.

현재 프로그램 복제 문제를 검출하기 위해 다양한 방법이 제시되고 있다[3-9]. 하지만 기존의 검출 방법은 복제로 의심되는 프로그램은 하나의 원본 프로그램에서 복제되었다고 판단한다. 하지만 이러한 판단은 복제로 의심되는 프로그램이 여러 원본 프로그램으로부터 복제되었다면 복제를 검출하기는 어려워진다. 이 논문에서는 여러 원본 프로그램으로부터 모듈 단위 복제를 통해 하나의 프로그램으로 복제하는 문제를 부분 복제 문제라 정의한다.

이 논문에서 부분 복제 문제를 해결하는 방법으로 프로그램 간의 유사한 구간을 이용하는 방법을 제시한다. 이를 위해 이 논문은 기존의 프로그램 유사도 분석 도구인 SOVAC(Software Verification and Analysis Center)[4]을 이용해서 프로그램 그룹 간의 유사한 구간 추출한다[3-9]. 그리고 추출된 프로그램 그룹 간의 유사한 구간 정보를 이용해서 부분 복제 문제가 있는 프로그램인지를 판단한다.

이 논문 2장에서는 관련 연구로서 프로그램 복제와 관련된 연구와 기존의 프로그램 유사도 분석 도구인 SOVAC에 대해 다룬다. 그리고 3장에서는 실제 프로그램 소스 코드를 이용해서 프로그램 부분 복제에 대해 설명한다. 이후 4장에서 부분 복제 문제를 해결하는 방법에 대해 언급하고 5장에서 결론을 맺는다.

2. 관련연구

프로그램 복제 검출을 위한 연구는 이미 다수 존재한다[3-9]. 첫 번째 방법은 문자열 비교를 통한 검출방법이다[3]. 해당 방법은 프로그램 소스코드를 문자열로 취급해서 같은 문자열을 찾는 알고리즘을 이용해서 유사한 구간을 검출하는 방법이다. 이 방법은 문장 삽입이나 제거, 변수명 변경 등과 같은 방법에 취약하다.

두 번째는 프로그램을 컴파일 과정에서 나오는 토큰으로 변환시켜 비교하는 방법이다[3,4]. 해당 방법은 문자열 비교와 달리 변수명 변경하여 문제는 해결할 수 있다. 하지만 문장 삽입이나 제거와 같은 방법에는 취약하다.

세 번째는 프로그램을 AST(Abstract Syntax Tree)로 변환해서 비교하는 방법이다. 해당 방법은 토큰 비교보다 프로그램의 구조를 반영해서 비교하는 방법이다[3]. 하지만 AST 구조의 변경이 발생하는 문장 삽입이나 제거와 같은 방법에는 취약하다.

네 번째는 프로그램을 PDG(Program Dependency Graph)로 변환해서 비교하는 방법이다[3,8]. 해당 방법은 프로그램을 그래프로 표현하기 때문에 변수 간의 데이터

"본 연구는 지식경제부 산업원천기술개발의 일환으로 수행하였음. [10035185, 서버기반 SW서비스의 분할 실행 기술 개발]"

의존성이나 제어 흐름 정보를 반영해서 복제를 검출한다. 하지만 PDG는 그래프 표현이기 때문에 제어 변경에 취약하다. 또한 PDG를 비교하는 방법은 동형 그래프를 찾는 문제이기 때문에 검출시간이 오래 걸리는 취약점이 있다.

마지막으로 프로그램의 구조적 특성을 이용하는 방법이 있다[5-7, 9]. 해당 방법은 Tamada[9]등이 제안한 방법으로 프로그램이 가지는 고유의 구조적 특성을 추출해서 비교하는 방법이다. 해당 분류의 기법은 프로그램의 크기가 작은 프로그램에서 적용하지 못하는 취약점이 있다.

기존의 유사도 분석 도구인 SOVAC은 토큰을 이용해서 유사한 프로그램을 검출하는 방법을 기반으로 한다[4]. SOVAC은 단순히 어휘 분석기의 토큰을 추출하는 것이 아니라 프로그램의 실행 경로를 고려하는 방법인 프로그램 정적 추적 방법을 이용해서 토큰 기반이지만 제어 흐름을 고려하는 방법을 제공한다. 그리고 토큰을 비교하는 방법으로 정렬(alignment)을 사용하는데 지역 정렬뿐만 아니라 토큰의 빈도수를 고려하기 위해 TF/IDF 방법을 이용해서 정렬하는 비교 방법을 제공한다.

3. 부분 복제 정의

부분 복제란 프로그램 복제 문제 중 복제로 의심 가는 프로그램이 하나의 원본 프로그램에서 복제되는 것이 아니라 다양한 원본 프로그램으로부터 모듈단위의 복제가 이루어지는 경우를 의미하는 것으로 정의한다. 이 문제는 프로그래밍 과제에서 프로그램 간의 인터페이스가 유사하거나 인터페이스가 지정된 프로그램일 때 쉽게 발생 할 수 있다.

이 문제는 기존의 프로그램 복제 검출 방법에서는 검출하기 어렵다. 그 이유는 기존의 검출 방법은 일반적으로 하나의 원본 프로그램에서 복제된 프로그램을 검출하는 것이다. 하지만 부분 복제 문제는 여러 원본 프로그램에서 모듈 단위로 복제하게 되면 각각의 원본 프로그램과 의심이 가는 프로그램의 유사도를 비교해보면 계산되는 유사도 값은 낮게 측정되기 때문에 기존의 검출 방법으로는 부분 복제 문제는 검출하기 어렵다.

아래의 프로그램 예제는 실제 부분 복제 문제를 보이기 위한 프로그램이다. 각각의 프로그램은 모두 문자열을 입력으로 받아 숫자로 변환하는 프로그램이다. 그리고 각각의 프로그램은 같은 행동을 수행하지만 각기 다른 코드로 작성되었기 때문에 원본 프로그램 간에는 복제 문제가 없는 코드이다.

아래의 <표 1>의 아스키코드를 이용하는 프로그램 A는 입력받은 문자를 숫자로 변환하기 위해서 라인 1~3에서 함수 `Convctoi()`를 선언한다. `Convctoi()`는 아스키코드를 이용해서 문자로 들어온 변수 c의 값을 받아 정수형 값으로 변환한 뒤 result에 할당해서 반환한다. 그리고 함수 `main()`에서는 buffer 변수에 문자열을 입력받고 자릿수 계산을 위해 입력받은 buffer의 값을 Rbuffer 변수를 이용해 입력된 값을 뒤집어서 저장한다. 이후 Rbuffer 배열의 각 원소 값을 `Convctoi()`를 통해 정수형으로 변경하고 자릿수에 맞게 계산해서 결과에 더한 뒤 최종값을 정수로 출력하는 프로그램이다.

<표 2>의 스위치문을 이용하는 프로그램 B는 입력받은 문자를 숫자로 변환하기 위해서 라인 1~8에서 함수 `Convctoi2()`를 선언한다. `Convctoi2()`는 인자로 받은 문자 c의 값을 스위치문을 통해 문자를 비교해서 해당 문자에 맞는 정수 값으로 변환한 뒤 result에 저장해서 반환한다. 그리고 함수 `main()`에서는 buffer 변수에 문자열을 입력받는다. 그리고 입력받은 buffer의 값을 자릿수에 맞게 계산하기 위해서 buffer의 원소를 방문하는 순서를 뒤에서부터 방문하는 것으로 buffer 배열의 각 원소 값을 `Convctoi2()`를 통해서 정수형으로 변경하고 자릿수에 맞게 계산해서 결과에 더한 뒤 최종값을 정수로 출력하는 프로그램이다.

<표 1> 아스키코드를 이용하는 원본 프로그램 A

```
 1:  int Convctoi(char c) {
 2:      int result = c - '0';
 3:      return result; }
 4:  int main() {
 5:      ...
 6:      cin>>buffer;
 7:      for(int i = 0; i < size; i++ )
 8:          Rbuffer[size-1-i] = buf[i];
 9:      for(int i = 0; i < size; i++)
10:          temp = Convctoi(Rbuffer[i]);
11:          result = result + temp * ndigits;
12:          ndigits = ndigits * 10; }
13:      cout<<"결과는 "<<result; }
```

<표 2> 스위치문을 이용하는 원본 프로그램 B

```
 1:  int Convctoi2(char c) {
 2:      int result = 0;
 3:      switch(c){
 4:      case '1': result = 1; break;
 5:      ...
 6:      case '0': result  = 0; break;
 7:      default: break; }
 8:      return result; }
 9:  int main() {
10:      ...
11:      cin>>buffer;
12:      for(int i = size-1; i >= 0; i-- ) {
13:         temp = Convctoi2(buffer[i]);
14:         result = result + temp * ndigits;
15:         ndigits = ndigits * 10; }
16:      cout<<"결과는 "<<result; }
```

<표 3>의 프로그램 C는 프로그램 A와 B를 이용해서 부분 복제가 이루어진 프로그램이다. <표 3>의 라인 1~3

은 프로그램 A로부터 문자를 입력으로 받아 정수를 반환하는 함수인 Convctoi()를 복제한 모듈이다. 그리고 라인 6~11의 경우 프로그램 B로부터 자릿수를 계산하기 위한 모듈을 복제한 경우이다.

<표 4>는 위의 세 프로그램 A, B, C를 대상으로 기존의 유사도 측정 도구인 SOVAC을 이용해서 유사도를 계산한 결과이다. <표 4>에서 P1과 P2는 측정 대상 프로그램의 이름을 의미하며 유사도는 두 프로그램 간이 얼마나 유사한지를 나타내는 척도로써 0~100% 사이의 값으로 나타난다. 그리고 $P1_S$와 $P1_E$는 대상 프로그램 P1의 어떤 부분이 P2와 유사한지를 나타내는 정보로써 $P1_S$는 P1에서 유사한 구간의 시작 지점을 의미하며, $P1_E$는 P1에서 유사한 구간이 끝나는 지점을 의미한다. 이때 유사한 구간의 정보는 소스코드의 라인을 의미하는 것이 아니라 SOVAC에서 프로그램을 토큰으로 변경했을 때 토큰이 일치하는 구간을 의미한다. 그리고 $P2_S$와 $P2_E$는 $P1_S$와 $P2_E$와 마찬가지로 프로그램 P2의 어떤 부분이 P1과 유사한지를 나타내는 정보이다.

<표 3> 원본 프로그램 A, B를 복제한 프로그램 C

```
 1:  int Convctoi(char c) {
 2:      int result = c - '0';
 3:      return result; }
 4:  int main() {
 5:      ...
 6:      cin>>buffer;
 7:      for(int i = size-1; i >= 0; i-- ) {
 8:         temp = Convctoi(buffer[i]);
 9:         result = result + temp * ndigits;
10:         ndigits = ndigits * 10; }
11:      cout<<"결과는 "<<result; }
```

<표 4> 프로그램 A, B, C를 대상으로 한 SOVAC 결과

P1	P2	유사도(%)	$P1_S$	$P1_E$	$P2_S$	$P2_E$
A	B	31.72	33	47	59	73
A	C	58.32	24	47	16	39
B	C	56.46	0	23	0	23

위의 <표 4>의 내용을 살펴보면 SOVAC은 부분 복제된 프로그램 C의 유사도를 살펴보면 프로그램 A와는 58.32% 유사하다고 판단하며 프로그램 B와는 56.46% 유사하다고 판단한다. 해당 유사도 수치는 실제 유사하다고 판단하는 기준에 미치지 못하는 수치로서 SOVAC에서는 프로그램 C는 프로그램 A와 프로그램 B와 유사하다고 판단하지 못한다. 이처럼 실제 복제된 프로그램이지만 기존의 유사도 검출 방법에서 검출하지 못하는 방법이 부분 복제 문제이다.

4. 검출방법

이 논문에서는 위의 3장과 같은 부분 복제 문제를 해결하기 위해서 프로그램 간의 유사한 구간을 이용하는 방법을 제안한다. 제안 방법은 기존의 유사도 정보를 이용하는데 실제 유사하다고 판단되지 않지만 일정 이상의 유사도가 높은 프로그램을 대상으로 부분 복제 문제를 검사한다. 기존의 SOVAC에서는 두 프로그램 A, B와 B, C의 유사한 구간을 아래 <식 1>과 같이 정의하고 있다[4]. <식 1>은 두 프로그램 간의 유사한 구간을 쌍들의 집합으로 정의하며 한 쌍에서 첫 번째 값은 프로그램 A의 토큰 값이며 두 번째 값은 프로그램 B의 토큰 값을 의미한다. 각 쌍의 의미는 두 토큰은 유사한 토큰이라는 것을 의미한다.

$$align(A,B) = < (a_1,b_1),(a_2,b_2),\dots,(a_n,b_n) >$$
$$align(B,C) = < (b_1,c_1),(b_2,c_2),\dots,(b_m,c_m) >$$

<식 1> 프로그램 (A, B), (B, C)의 유사한 구간

아래의 <식 2>는 세 프로그램 간에 유사한 구간이 <식 1>과 같을 때 부분 복제라 판단하는 방법을 의미한다. <식 2>에서 함수 *fst()*는 각 쌍에서 첫 번째 값을 가져오는 함수이며, 함수 *snd()*는 각 쌍에서 두 번째 값을 가져오는 함수이다. 그리고 ε은 판단의 정확성을 높이기위해 오차 범위를 고려하기 위한 변수로써 0~1 사이의 값을 가진다. <식 2>는 두 유사한 구간에서 공통되는 프로그램인 B의 각 유사한 구간에 대해 교집합 연산을 수행했을 때 결과의 원소의 수가 오차 범위이내인 경우 부분 복제가 발생했다고 판단한다.

$$0 \le |snd(align(A,B)) \cap fst(align(B,C))| \le 1-\epsilon$$

<식 2> 프로그램 A, B, C의 부분 복제 판단 방법

아래 <표 5>는 <식 2>를 기반으로 부분 복제를 검출하기 위한 알고리즘이다. 아래의 알고리즘은 입력으로 프로그램 P1에 대한 유사 구간 집합 리스트를 받는다. 그리고 intersection() 함수를 이용해서 P1 리스트의 원소를 대상으로 하나의 유사한 구간이 다른 모든 유사한 구간과 겹치는지 검사한다. 그리고 <표 6>에서 정의된 intersection() 함수는 두 유사구간의 시작점과 끝점을 받아서 겹쳐지는 구간의 토큰의 개수를 길이로 봐서 0~1 사이의 값으로 나타내는 함수이다. intersection() 함수는 두 구간이 겹치지 않는다면 0을 반환하고, 두 구간이 완전히 일치한다면 1을 반환한다. 그 외에는 두 구간이 겹쳐지는 영역의 길이를 계산한다. 그리고 계산된 길이를 이용해서 두 구간 중 긴 구간의 겹쳐지지 않는 나머지 구간을 0~1 사이의 값으로 나타내어 반환하는 함수이다. 그리고 <표 5>의 알고리즘에서 계산된 두 구간이 겹치는 수치가 1 - ε 이하인 경우 해당 코드는 부분 복제 문제가 발생했다고 보고하고 그 이외는 부분 복제 문제가 발생하

지 않은 상태로 정의함으로 부분 복제 문제를 검출한다. 그리고 제안 기법의 알고리즘은 하나의 구간에 대해 다른 모든 구간이 겹치지는 검사하기 때문에 $O(n^2)$으로 동작한다.

아래 <표 7>은 대상 프로그램 A, B ,C에 <표 5>의 알고리즘을 적용해 나온 결과이다. <표 7>은 50% 이하의 유사도 집합은 포함하지 않았으며 ε값은 0.3으로 적용 시킨 결과이다. 프로그램 A의 경우 B, C에 대해 겹쳐지며 프로그램 B의 경우 유사도가 50% 이상인 유사한 구간 집합의 개수가 하나이기 때문에 결과가 나오지 않는다. 그리고 프로그램 C의 경우 A, B와 유사한 구간의 값은 8이 나오기 때문에 겹쳐지는 값으로 0.67을 반환해서 부분 복제가 발생했다고 알려준다.

<표 5> 부분 복제를 검출하기 위한 알고리즘

```
P1 <- 유사한 구간 집합 List
while ( P1.tail ≠ nil ) {
    temp := P1.tail;
    while( temp.tail ≠ nil ) {
        a := intersection(temp, P1);
        if( a ≤ 1 - ε )
            Report Occurred Clone
        else
            nil;
        temp := temp.tail;
    }
    P1 := P1.tail;
}
```

<표 6> intersection 함수 정의

```
intersection :: (pair, pair) -> double
intersection (A, B) :=
    if (Eq A B)
       return 1.0;
    else if( length(A) < length(B) )
       intersection(B, A);.
    else {
       diff = overlap( A, B );
       return (length(A) - diff) / length(A);
```

<표 7> 프로그램 A, B, C 대상의 복제 판단 결과

P1	P2	판단값	결과
A	B,C	1	부분 복제 미발생
B	A,C	nil	부분 복제 미발생
C	A,B	0.67	부분 복제 발생

5. 결론

이 논문에서는 실제 프로그램을 이용해서 부분 복제 문제에 대해서 정의하였다. 이 문제는 하나의 원본 프로그램에서 복제하는 것이 아니라 여러 원본 프로그램에서 모듈 단위로 복제하는 것을 의미한다. 그리고 기존의 유사도 검출 기법에서 부분 복제 문제를 검출하지 못함을 보이고, 해당 문제를 해결하는 방법을 제안하였다. 제안 방법은 토큰을 기반으로 한 기존의 SOVAC의 정보 중 프로그램 간의 유사한 구간을 이용하는 방법으로 유사한 구간이 겹치지 않는다면 부분 복제로 판단한다.

향후 연구로는 토큰 기반의 검출 기법뿐만 아니라 다른 검출기법에 해당 방법을 적용해 결과를 관찰해볼 필요가 있다. 현재는 SOVAC의 정보를 이용해서 실험해 보았지만 이후 구조적 특성을 이용한 검출 방법 등 다양한 방법을 이용해 부분 복제 문제를 검출해 볼 예정에 있다. 그리고 제안 기법은 부분 복제 검출 시 비교 대상이 늘어남에 따라 비교 횟수가 많이 늘어나는 문제점을 가지기 때문에 비교 횟수를 줄이는 방법에 관한 연구가 추가로 필요하다.

참고문헌

[1] M. Joy, M. Luck, “Plagiarism in programming assignments”, IEEE Transactions on Education, 42(2):129{133, 1999.

[2] L. Prechelt, G. Malpohl, M. Philippsen, “Finding plagiarisms among a set of programs with JPlag”, J. UCS, 8(11):1016, 2002.

[3] C. Roy, J. Cordy, “A survey on software clone detection research”, Queenïs School of Computing TR, 541:115, 2007.

[4] 지정훈. “적응적 서열 정렬 기법을 이용한 프로그램 유사도 분석 프레임워크”, 박사학위논문, 부산대학교, 2010.

[5] X. Wang, Yoon.-Chan Jhi, S. Zhu, and P. Liu, “Behavior based software theft detection”, Proceedings of the 16th ACM conference, 2009.

[6] P Chan, L Hui, S Yiu, “Dynamic Software Birthmark for Java Based on Heap Memory Analysis”, Communications and Multimedia Security, 2011.

[7] H Park, H Lim, S Choi, T Han, “Detecting Common Modules in Java Packages Based on Static Object Trace Birthmark”, The Computer Journal, Volume 54, Issue 1, pp.108-124, 2011.

[8] C Liu, C Chen, J Han, PS Yu, “GPLAG: detection of software plagiarism by program dependence graph analysis”, Proceedings of the 12th ACM SIGKDD, 2006

[9] H. Tamada, M. Nakamura, A. Monden, and K. Matsumoto, “Java birthmark — Detecting the software theft”, IEICE Transactions on Information and Systems, E88-D, 9, pp.2148-2158, 2005.