논문 2020-2-1 http://dx.doi.org/10.29056/jsav.2020.12.01

목적 코드에서 유사도 검출과 그 도구의 설계

유장희*†

Similarity Detection in Object Codes and Design of Its Tool

Jang-Hee Yoo*

요 약

컴퓨터 프로그램의 표절 또는 복제에 대한 유사도 검출은 구현에 사용된 프로그래밍 언어, 분석 대상 코드의 종류에 따라 각기 다른 감정 방법과 도구가 필요하다. 최근 들어 증가하고 있는 내장형 시스템의 목적 코드에 대한 유사도 감정은 원시 코드와 비교해 더욱 복잡한 과정 및 고도의 기술과 함께 상당한 자원을 요구하고 있다. 본 연구에서는 목적 코드를 대상으로 역 어셈블리 기법의 적용과 같은 역공학 방법을 이용하여, 목적 코드의 어셈블리어 변환을 통한 어셈블리어 레벨에서의 함수 단위 유사도 감정 방법에 관하여 연구하였다. 그리고, 어셈블리어 레벨에서의 유사도 비교를 위해 코드의 구문분석을 통한 명령어 및 오퍼랜드 테이블을 생성하고, 이를 대상으로 유사도를 검출할 수 있는 도구의 설계에 관하여 기술하였다.

Abstract

The similarity detection to plagiarism or duplication of computer programs requires a different type of analysis methods and tools according to the programming language used in the implementation and the sort of code to be analyzed. In recent years, the similarity appraisal for the object code in the embedded system, which requires a considerable resource along with a more complicated procedure and advanced skill compared to the source code, is increasing. In this study, we described a method for analyzing the similarity of functional units in the assembly language through the conversion of object code using the reverse engineering approach, such as the reverse assembly technique to the object code. The instruction and operand table for comparing the similarity is generated by using the syntax analysis of the code in assembly language, and a tool for detecting the similarity is designed.

한글키워드 : 프로그램 유사도, 프로그램 표절, 목적 코드 비교, 소프트웨어 감정평가

keywords : program similarity, program plagiarism, object code comparison, software appraisal

1. 서 론

컴퓨터 프로그램의 유사도 분석 및 검출에 관한 연구는 컴퓨터 교육과정에서 수강생 프로그램의 표절을 찾아내기 위해 시작되었다고 한다[1]. 가장 단순한 형태의 프로그램 표절 탐지에 대한 회피를 위해 변수 또는 함수명 변경, 코드 블록들 위치의 재배치, 공백 또는 주석의 변경 등의 방법을 사용하였다. 이러한 표절 탐지를 위하여

* 한국전자통신연구원 인공지능연구소
† 교신저자: 유장희 (email: jhy@etri.re.kr)
접수일자: 2020.11.17. 심사완료: 2020.12.02.
게재확정: 2020.12.21.

다양한 종류의 도구들이 개발되었으며[2], 잘 알려진 도구들로는 MOSS[3], JPlag[4], Plague[5], SIM[6], YAP3[7] 등이 있다. 국내에도 한국저작권위원회에서 원시 코드의 복제/표절 감정작업을 돕기 위해 개발한 exEyes가 있다[8].

최근에는 개발업체에서도 다양한 경로로 기술유출 또는 복제가 빈번히 발생하고 있어, 이에 대한 감정 요구가 증가하는 추세에 있다. 감정의 목적물은 대부분 동일 프로그래밍 언어로 구현된 원시 코드가 일반적 경우이다[8]. 감정을 위해 exEyes 등과 같은 앞서 기술된 다양한 종류의 원시 코드에 대한 표절 탐지 도구를 이용하거나 코드 리뷰 등을 통하여 논리적 또는 물리적 유사도 감정에 의한 표절 여부를 판단할 수 있다.

한편, 각종 정보기기 및 모바일기기와 같은 IoT 또는 내장형 시스템(embedded system)이 증가함에 따라 감정 목적물이 원시 코드(source code)가 아닌 ROM에 저장된 목적 코드(object code) 또는 실행 프로그램의 형태로 제공되는 경우가 증가하고 있다[9][10]. 이러한 경우는 역공학(reverse engineering) 방법을 적용하여 목적 코드나 실행 프로그램을 어셈블리어로 변환하여, 변환된 결과물을 대상으로 유사도 감정을 수행하는 것이 일반적인 방식이다. 그러나 어셈블리어 레벨에서의 유사도 감정은 코드의 복잡도와 방대함으로 인해 고도의 기술이 요구되며, 많은 어려움이 있을 수 있다. 본 논문에서는 역공학 방법에 따른 목적 코드에서의 유사도 검출 방법 및 이를 위한 도구의 설계에 관하여 기술하였다.

2. 목적 코드에서 유사도 분석 방법

현재의 국내 저작권법은 목적 코드에 대해서도 저작권을 부여하여, 컴퓨터 프로그램저작물로 인정하는 것이 통설과 판례로 나타나고 있다[9]. 일반적으로 컴퓨터 프로그램의 유사도 감정은 물리적 유사도 분석과 논리적 유사도 분석으로 분류할 수 있다. 또한, 제공되는 목적물에 따라 동일 언어 간 유사도 감정, 이종 언어 간 유사도 감정, 목적 코드 간 유사도 감정 등으로 분류할 수 있다. 목적 코드의 유사도 감정은 일반적으로 복제 또는 표절을 의심받는 측에서 원시 코드의 제출을 거부하거나 감정의 특성상 정보기기와 같은 제품 내부에 저장된 목적 코드만으로 감정해야 하는 경우이다[10]. 즉, 양측의 목적 코드에 대하여 역 어셈블리 기법에 따른 어셈블리어 코드를 확보함으로써, 동일 언어 간 유사도 감정을 수행하는 방식이다. 그러나 목적 코드에 대한 어셈블리 기법은 복제된 원시 코드의 리스트 작성이나 기능의 유사성 분석에는 많은 제약사항이 있다.

2.1 목적 코드 유사도 분석 과정

그림 1은 목적 코드에 대해 기본적으로 적용될 수 있는 유사도 분석 과정 및 방법을 나타내고 있다[11]. 일반적으로 역 어셈블리 방법으로 확보된 어셈블리어는 고수준의 프로그램 언어와 비교해 논리적 흐름의 분석 등에 대한 난이도 및 코드의 분량이 상당히 증가하게 된다. 따라서 아무런 도구 없이 수작업에 의한 동일함수 비교 및 유사 함수에 대한 목록 작성은 상당한 시간과 노력을 요구한다. 더불어 원시 코드의 컴파일 과정에서 적용하는 옵션에 따라 다른 형태의 코드가 생성될 수 있어, 표절된 원시 코드의 목록 작성은 부분적으로 수행될 수밖에 없다.

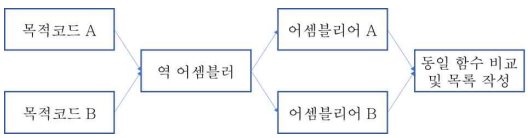

그림 1. 목적 코드의 유사도 분석
Fig. 1. Similarity Analysis in Object Codes

또한, 목적 코드에 대한 역 어셈블리 도구를 이용하여 생성된 어셈블리어 코드의 함수 이름들은 그 특성상 주소 번지로 표시된다. 이들 주소 번지로 표시된 함수의 이름들은 어느 한쪽의 원시 코드가 확보된다면 이와 대응되는 실제 함수의 이름들을 추출할 수 있다. 이는 Visual C/C++ 등의 경우 릴리즈 모드가 아닌 디버깅 모드를 사용하여 생성된 목적 코드에 대한 어셈블리어의 경우는 함수의 실제 이름을 유지하고 있는 특성을 이용하면 가능하다. 즉, 구현된 원시 코드를 릴리즈 모드와 디버깅 모드 각각으로 컴파일하여, 릴리즈 모드에서 주소 번지로 표시된 함수 이름과 대응되는 실제 이름을 디버깅 모드로 컴파일된 어셈블리어에서 추적하는 방법을 이용하면 비교적 쉽게 어셈블리어 코드에서 동일함수로 검출된 목록에 대응되는 함수의 실제 이름을 찾아낼 수 있게 된다[11].

2.2 실행 파일의 어셈블리어 코드 생성

목적 코드에 대한 유사도 검출의 첫 번째 단계는 역 어셈블리 기법에 따른 어셈블리어 코드의 생성이다. 윈도우 시스템에서 자주 사용되는 도구로는 마이크로소프트 Visual Studio에서 제공되는 DumpBin과 SmidgenSoft에서 제공되는 PEBrowse Professional Interactive[12] 라는 도구가 있으나, IDA Pro[13]가 가장 유용한 도구로 많이 사용되고 있다. Win32 프로세스의 경우 가상주소 공간 내에서 컴파일러나 어셈블리어가 최종적으로 생성하는 목적 코드는 "텍스트 섹션" 또는 ".text" 섹션에 존재한다[14].

그림 2는 컴퓨터 프로그램이 원시 코드 형태에서 컴파일러에 의하여 기계어로 번역되어 최종적으로 실행 프로그램(PE: Portable Executable) 섹션에 저장되게 되는 과정을 간단히 도시한 것이다. 일반적으로 텍스트 섹션에 저장되는 목적 코드는 역 어셈블리 도구를 이용하여 어셈블리어로 다시 변환할 수 있다. 그림에서 좌측 블록의 코드들은 우측 블록의 기계어 코드와 대응되는 어셈블리어 코드를 나타낸다. 그림에서 보는 것과 같이 어셈블리어는 push, mov, sub, lea, rep, call 등의 제한적이고 비교적 단순한 명령어들과 esi, 0, eax, [msg] 등의 오퍼랜드들로 구성된다.

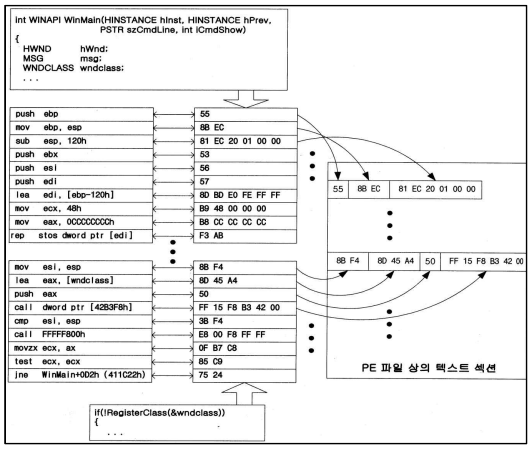

그림 2. PE 텍스트 섹션의 생성 [14]
Fig. 2. Generation of PE Text Section [14]

따라서, 어셈블리어 코드에 대한 논리적 흐름에 대한 분석은 매우 어려울 수 있으나 코드의 물리적 구조 및 유사도는 비교적 쉽게 검출될 수 있다. 즉, 각각의 목적 코드는 역 어셈블리 도구를 사용하면 그림 2에서와 같은 명령어와 오퍼랜드들로 구성된 어셈블리어 코드로 변환되며, 이로부터 유사도 검출이 가능한 어셈블리어 코드의 동일 언어 간의 유사도 감정이 가능하게 된다.

3. 어셈블리어 코드에서 유사도 검출

역 어셈블리 도구에 의해 생성된 어셈블리어(assembly language) 코드의 유사 비교는 기본적으로 함수 단위의 비교가 적합하다. 그러나 유사

도 비교를 위한 두 코드 상호 간에 비교할 대상 함수를 찾는 것 또한 어려운 작업이다. 따라서, 원시 코드와 비교해 상당히 많은 양의 어셈블리어 코드에 대한 함수 단위 비교를 위해서는 이를 위한 도구의 개발 또한 필요하게 된다.

3.1 역 어셈블리어 코드의 유사도

그림 3은 실행 파일에 대해 역 어셈블리 도구인 IDA[13]를 이용하여 생성한 어셈블리어 코드의 예이다. 그림에서 볼 수 있듯이 양쪽 어셈블리어 코드는 주소와 관련된 부분을 제외하면 모두 같다는 것을 쉽게 알 수 있다. 단순한 함수에 대하여는 수작업에 의해서도 함수 간 유사도의 검출이 가능하다. 그러나, 수작업으로 전체 프로그램에서 함수 간 유사도 비교는 많은 어려움과 노력을 요구한다. 이는 역 어셈블리어에 의하여 생성된 코드는 그 양이 상당히 많으며, 일반적으로 각각의 코드에 나타나는 유사 함수들의 위치가 순차적이지 않은 데 기인한다.

```
.text:10001550 ; ------ S U B R O U T I N E ------
.text:10001550-
.text:10001550
.text:10001550 sub_10001550   proc near
; DATA XREF: .data:off_10018C80o
.text:10001550
.text:10001550 arg_0  = dword ptr  4
.text:10001550 arg_4  = dword ptr  8
.text:10001550 arg_8  = dword ptr  0Ch
.text:10001550
.text:10001550    mov    eax, [esp+arg_8]
.text:10001554    mov    ecx, [esp+arg_4]
.text:10001558    push   eax
.text:10001559    mov    eax, [esp+4+arg_0]
.text:1000155D    xor    edx, edx
.text:1000155F    push   ecx
.text:10001560    mov    dx, [eax+10h]
.text:10001564    add    eax, 12h
.text:10001567    push   edx
.text:10001568    push   eax
.text:10001569    call   sub_10002070
.text:1000156E    add    esp, 10h
.text:10001571    retn
.text:10001571 sub_10001550  endp
```

```
.text:10004F00 ; ------ S U B R O U T I N E ------
.text:10004F00
.text:10004F00
.text:10004F00 sub_10004F00   proc near
; DATA XREF: .rdata:off_1000F930o
.text:10004F00
.text:10004F00 arg_0  = dword ptr  4
.text:10004F00 arg_4  = dword ptr  8
.text:10004F00 arg_8  = dword ptr  0Ch
.text:10004F00
.text:10004F00    mov    eax, [esp+arg_8]
.text:10004F04    mov    ecx, [esp+arg_4]
.text:10004F08    push   eax
.text:10004F09    mov    eax, [esp+4+arg_0]
.text:10004F0D    xor    edx, edx
.text:10004F0F    push   ecx
.text:10004F10    mov    dx, [eax+0Eh]
.text:10004F14    add    eax, 10h
.text:10004F17    push   edx
.text:10004F18    push   eax
.text:10004F19    call   sub_10001AF0
.text:10004F1E    add    esp, 10h
.text:10004F21    retn
.text:10004F21 sub_10004F00  endp
```

그림 3. 유사 함수의 어셈블리어 코드
Fig. 3. Assembly Language for Similar Functions

또한, 어셈블리어 코드의 비교로 검출된 동일 함수 목록은 목적 코드의 특성상 그림에서와같이 실제 함수의 이름이 아닌 서브루틴의 주소 번지로 표시된다. 따라서, 주소 번지로 표시된 동일 함수의 이름들은 원시 코드에서의 실제 함수 이름으로 대치가 필요하게 된다. 이를 위하여는 앞서 기술했듯이 적어도 한쪽의 원시 코드를 이용하여야 하는데, 원시 코드를 릴리즈 모드가 아닌 디버깅 모드로 컴파일하는 경우 함수의 이름은 주소 번지로 변환이 되지 않고 원시 코드에서의 본래 이름을 유지하는 특성을 이용하여 수행할 수 있다. 따라서, 주소 번지로 표시된 동일함수 목록과 디버깅 모드에서의 어셈블리어 코드의 함수 이름을 비교하면 유사도가 확인된 함수의 실제 목록을 추출할 수 있게 된다.

3.2 함수 연관도를 이용한 유사도 분석

함수 연관도는 함수 간의 호출 위치나 서브루틴의 호출 관계를 나타내는 물리적 또는 논리적 유사도 비교의 중요한 도구로 전체 프로그램의 구조를 직관적으로 나타내는 방법이다. 원시 코드의 분석에 사용할 수 있는 연관도 분석 도구로는 Source Insight[15] 등이 있으며, 역 어셈블리 분석 및 목적 코드의 연관도 분석을 위한 도구로는 앞서 언급한 IDA[13] 등이 있다. 함수 연관도 분석을 위해서는 먼저 양측 비교 대상의 목적 코드 각각에 대한 연관도를 추출하여야 한다. 그리고 적어도 한쪽의 원시 코드에 대응하는 목적 코드의 함수 연관도 작성은 목적 코드의 각 함수와 원시 코드의 각 함수 상호 간에 어떻게 사상 되는지 분석하는 것이 중요한 요소이다[11].

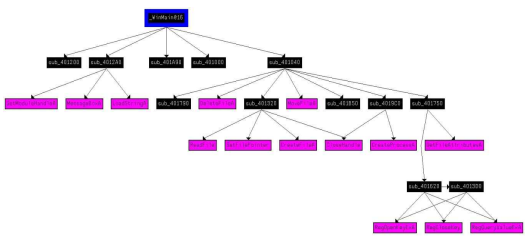

그림 4. 목적 코드의 함수 연관도 예
Fig. 4. Function Call Graph in Object Code

그림 4는 IDA를 이용하여 추출한 목적 코드의 함수 연관도 예이다. 목적 코드의 함수 연관도를 이용하여 원시 코드의 함수 연관도 상에서 대응되는 함수명을 추출하기 위해서는 목적 코드와 원시 코드가 동일 버전이어야 1:1 사상에 대한 연관도를 작성할 수 있다. 함수 연관도의 유사도 분석은 다음과 같은 과정으로 수행할 수 있다.

1) 각각의 목적 코드 주요 루틴에 대한 함수 연관도를 역 어셈블리 기법으로 생성
2) 생성된 어셈블리어 코드의 유사도 비교를 통하여, 검출된 동일함수 목록을 비교 대상 목적 코드의 함수 연관도에서 탐색 및 추출
3) 원시 코드가 존재하는 어셈블리어의 실제 함수명을 추출하여, 목적 코드의 함수 연관도와 비교를 통해 대응되는 실제 함수명으로 대체

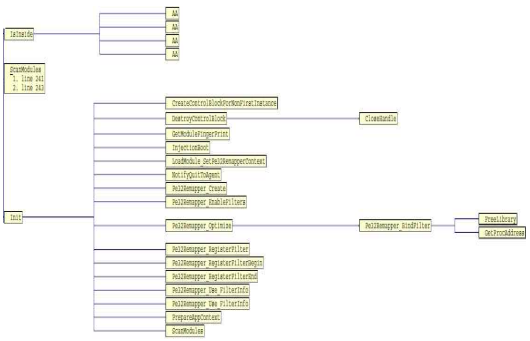

그림 5. 원시 코드의 함수 연관도 예
Fig. 5. Function Call Graph in Source Code

목적 코드와 원시 코드로부터 생성된 두 개의 함수 연관도를 상호 비교하여, 원시 코드에서 함수들과 목적 코드의 함수 간에 1:1 대응 관계의 분석을 시도할 수 있다. 이러한 대응 관계의 분석은 목적 코드에서 실질적으로 호출된 함수들의 이름을 추출하여 유사도 감정 대상 프로그램 상호 간의 논리적 구조 및 호출 함수를 비교하는데 유용하게 사용될 수 있다. 그림 5는 원시 코드의 함수 연관도의 예를 나타낸 것이다.

4. 유사도 검출 도구의 설계

어셈블리어는 비교적 단순하고 한정된 명령어(instruction)들과 오퍼랜드(operand)들로 구성된다. 그러나 수작업에 의한 어셈블리어 레벨의 유사도 감정은 앞에서 기술한 다양한 이유로 상당한 시간과 고도의 기술을 요구한다. 따라서, 어셈블리어 레벨에서의 유사도 감정을 위해서는 명령어와 오퍼랜드의 토큰(token) 분류에 의한 테이블의 생성과 유사도를 자동으로 비교할 수 있는 도구의 개발이 필요하다. 또한, 이를 위한 유사도 감정 항목에 대한 분류가 필요하다.

4.1 어셈블리어 유사도 검출 도구 설계

함수 단위 비교를 위해서는 먼저 명령어와 오퍼랜드들을 토큰으로 분류한 후 함수마다 명령어와 오퍼랜드 테이블을 생성하여, 이들에 대한 비교를 수행할 수 있다. 표 1은 유사도 비교를 위해 요구되는 각각의 함수들에 대한 명령어 및 오퍼랜드 테이블의 예를 나타내고 있다. 테이블의 구성은 그림 3의 어셈블리어 코드의 스캔 순서에 의한 순차적 인덱스(index)와 토큰 분류에 의한 명령어, 오퍼랜드 1, 오퍼랜드 2, 그리고 오퍼랜드에 포함된 오프셋(offset)과 명령어의 출현 순서에 대한 정보로 구성할 수 있다.

표 1. 명령어 및 오퍼랜드 테이블의 예
Table 1. Example of Instruction and Operands

Index	Instruction	OP 1	OP 2	Offset	Order
1	add	eax	12h	12h	8
2	add	esp	10h	10h	12
3	call	sub_1	null	null	11
4	mov	exa	esp	arg_8	1
-	-	-	-	-	-
12	push	eax	null	null	3
13	xor	edx	edx	null	5

즉, 함수의 특징 테이블 간 비교를 효과적으로 수행하기 위하여 테이블의 구성은 명령어를 기준으로 정렬을 수행하며, 인덱스는 순차적으로 증가하는 명령어 배열의 순차 번지를 저장하고, OP 1 및 OP 2에는 오퍼랜드들을 저장한다. 그리고 오퍼랜드에 포함된 오프셋은 테이블의 Offset 영역에 분리 저장하여 비교 시 제외되도록 한다. 오퍼랜드와 함께 나타나는 레지스터 등을 범용, 오프셋, 세그먼트 등의 범주로 분류하여 추가적인 정보를 제공할 수도 있다. 테이블은 효율적 비교를 위하여 명령어를 기준으로 정렬되어 정렬 이전에 명령어의 위치에 대한 순서를 저장할 필요가 있으며, 이에 대한 정보는 Order 부분에 저장하게 된다. 각각의 함수별로 표 1과 같은 형식의 테이블을 생성하면 함수 간 명령어의 유사도, 오퍼랜드의 유사도, 명령어 순서의 유사도, 유효 라인 수 등 함수 간의 다양한 정보들에 대한 유사도를 정량화할 수 있다. 이때, 각각의 유사도에 대한 가중치는 달리 정의할 수 있다.

4.2 목적 코드 유사도 감정의 고려사항

앞서 기술한 바와 같이 목적 코드에 대한 유사도 감정은 역 어셈블리 도구를 사용하여, 어셈블리어 코드로 변환한 후 동일 언어 간 유사도 감정을 수행하게 된다. 그러나 역 어셈블리 도구의 한계로 동일 원시 코드가 존재하지 않는 경우의 표절 또는 복제된 원시 코드에 대한 목록 작성은 매우 제한적이다. 또한, 자동생성 코드나 공개 코드에 대한 탐지도 매우 제한적이다. 그럼에도 목적 코드의 유사도 분석은 작성된 프로그램 언어의 종류와 관계없이 모든 목적 코드를 어셈블리어 코드로 변환할 수 있어, 동일 언어 간 유사도 감정의 문제로 전환할 수 있다는 장점이 있다.

어셈블리어 코드를 이용한 물리적 유사도 감정 항목으로는 역 어셈블리 도구를 통해 얻을 수 있는 양쪽 목적 코드에 대응되는 파일의 크기, 함수 및 이름들의 개수, 동일함수의 개수 등과 같이 비교적 정량적으로 유사도를 산출할 수 있는 항목 간 비교 요소들을 찾을 수 있다. 논리적 유사도는 그 특성상 다소 주관적인 판단이 불가피하며, 정확한 근거의 제시에 의한 정량적 표현에는 어려운 부분이 있다. 일반적으로 원시 코드에 대하여는 다양한 분석 도구를 적용할 수 있어, 함수에 대한 논리적 구조를 비교적 쉽게 분석할 수 있다. 그러나 어셈블리어 코드에 대한 논리적 구조는 분석이 쉽지 않다.

따라서, 어셈블리어 코드에 논리적 유사도 감정에는 이에 대한 해독 능력 및 주관적 판단기준과 관련된 부분이 반영될 소지가 크다. 이러한 문제를 최소화하기 위해서는 함수에서 사용된 변수의 개수와 호출된 함수의 수 및 호출 위치 등 기능 및 구조적으로 유사하게 보이는 함수에 근거한 유사도 분석 항목이 필요하다. 물리적 유사도 분석에서는 완전히 같은 코드에 대하여는 분석을 쉽게 할 수 있으나 완전히 같지는 않지만, 함수 레벨에서의 기능 및 논리적으로 유사한 코드들의 검출이 가능할 것이다[11].

코드의 논리적 구조와 무관하지 않으나 비교적 정량적으로 분석하기 쉬운 방법으로는 각각의 파일들과 이들 각각에 대응하는 비교 대상 파일들에 대해 역 어셈블리 기법을 사용하여 생성한 어셈블리어 코드가 가지고 있는 전체 함수 및 이름의 개수에 대한 유사성을 비교할 수 있다. 전체 함수의 개수는 각각의 목적 코드에서 호출된 함수 중 컴파일러로부터 제공되는 라이브러리 함수의 개수를 제외한 기본적으로 원시 코드로 구현되어 변환된 함수의 개수를 의미한다. 또한, 이름들의 개수는 각각의 목적 코드에서 사용된 라이브러리 함수, 정규함수, 명령어, 스트링, 데이터 등의 이름들 개수를 의미한다.

프로그램 표절 또는 복제의 가장 확실한 증거

들은 함수 이름들의 유사성 또는 동일성 보다, 각각의 함수를 구성하고 있는 코드의 동일성 여부이다. 각 어셈블리어 코드에서 동일함수 비율을 분석하기 위해서는 앞에서 기술한 방법에 따른 함수 단위의 유사도 분석이 필요하다. 여기서 산출되는 결과는 호출된 함수의 개수 유사성과 같은 직관적 유사성과는 달리 정확한 분석을 통하여 산출된 사실적이고 객관적인 결과이다. 또한, 함수 연관도는 코드의 구조를 직관적으로 비교하는 방법으로 물리적 유사성을 내포하고 있는 논리적 유사성 비교의 중요한 도구이다.

5. 결 론

본 연구에서는 컴퓨터 프로그램의 표절 또는 복제도 검출을 위한 목적 코드의 유사도 검출 방법 및 그 도구의 설계에 관하여 기술하였다. 유사도 검출을 위한 대부분의 원시 코드 및 목적 코드들은 컴파일러와 역 어셈블리 방법에 의하여 어셈블리어 코드로 변환할 수 있다. 그리고 이렇게 생성된 어셈블리어 코드는 원시 코드보다 그 양은 많이 증가하나 비교적 간단한 형태의 구조를 유지하고 있어, 함수 레벨에서 양측 코드 간의 물리적 유사도 검출이 가능하다. 또한, 어느 한쪽의 원시 코드가 존재하는 경우 양측 프로그램의 동일함수 목록과 복제된 코드 부분을 추출할 수 있다. 본 연구에서는 이러한 유사도 감정 방법 및 이를 위한 도구의 설계에 관하여 기술하였다. 향후, 본 연구에서 제안한 어셈블리어 코드의 물리적 유사도 감정을 위한 도구의 구현 및 시험이 요구된다.

참 고 문 헌

[1] H. Berghel and D. Sallach, “Measurements of Program Similarity in Identical Task Environments”, *ACM SIGPLAN Notices*, 19(8), pp.65~76, Aug. 1984. https://doi.org/10.1145/988241.988245

[2] 김규식, 조성제, 우진운, “소스코드 유사도 측정 도구의 성능에 관한 비교연구”, 한국소프트웨어감정평가학회 논문지, 13권1호, pp.31-42, 2017.6. http://www.i3.or.kr/html/paper/2017-1/(4)2017-1.pdf

[3] A. Aiken, *MOSS: A System for Detecting Software Similarity*, Stanford University, USA, 2020. http://theory.stanford.edu/~aiken/moss/

[4] L. Prechelt, G. Malpohl, and M. Phlippsen, *JPlag:* Finding *Plagiarisms among a Set of Program*, Technical Report 2000-1, University of Karlsruhe, Germany, Mar. 2000. https://publikationen.bibliothek.kit.edu/542000/759910

[5] G. Whale, *Plague: Plagiarism Detection using Program Structure*, TR Vol.8805, Department of Computer Science, University of NSW, Kensington, Australia, 1988.

[6] D. Gitchell, N. Tran, “SIM: A Utility for Detecting Similarity in Computer Programs”, in *Proc. of 30th SIGCSE Technical Symposium on Computer Science Education*, New Orleans, USA, pp.266~270, May 1999. https://doi.org/10.1145/299649.299783

[7] M. Wise, “YAP3: Improved Detection of Similarities in Computer Program and Other Texts”, in *Proc. Of 27th SIGCSE Technical Symposium*, Philadelphia, USA, pp.130~134, 1996. https://doi.org/10.1145/236462.236525

[8] 한국저작권위원회, *2019년 SW감정인 역량제고 교육*, 자료집, 2019.12.

[9] 김시열, 주형락, “실행코드의 실질적 유사성 비교에 관한 저작권법 관점에서의 소고”, 한국소프트웨어감정평가학회 논문지, 12권2호, pp.15-24, 2016,12. http://www.i3.or.kr/html/paper/2016-2/(3)2016-1.pdf

[10] 이규대, “목적코드 파일의 유사도 측정 개선방안”, 한국소프트웨어감정평가학회 논문지, 8권1호, pp.41-48, 2012.6.
[11] 유장희, 실행파일의 유사도 감정평가 및 프로파일 작성방안, 컴퓨터프로그램보호위원회, 2007 SW 감정 워킹그룹 연구 결과 보고서, 2007.
[12] R. Osterlund, *PEBrowse Professional Interactive*, SmidgeonSoft, LLC., 2011. https://download.cnet.com/PEBrowse-Professional-Interactive/3000-2218_4-10445319.html
[13] S. Micallef, *IDA Plug-In Writing in C/C++*, 2020. https://www.hex-rays.com/products/ida/
[14] 이호동, *Windows 시스템 실행파일의 구조와 원리*, 한빛미디어, 2005. ISBN: 897914332x
[15] Source Dynamics, *Source Insight 4.0 User Manual*, 2020. https://www.sourceinsight.com/

저 자 소 개

유장희(Jang-Hee Yoo)

1988.02 한국외국어대학교 물리학과 졸업
1990.02 한국외국어대학교 전산학과 석사
2004.07 영국 University of Southampton 전자 및 컴퓨터과학 박사
1989.11-현재: 한국전자통신연구원 인공지능연구소 책임연구원
2005.09~현재: 한국저작권위원회 감정인, 감정위원
2007.03-현재: 과학기술연합대학원대학교 ICT전공 전임교수
2007.01~현재: 한국SW감정평가학회 이사
2014.01~현재: 경찰청 과학수사자문위원
2014.8~2015.8: University of Washington 방문학자
2018.3~2020.3: 국가지식재산위원회 전문위원

<주관심분야> 컴퓨터 비전, 인공지능, 생체인식, 휴먼모션분석, HCI 및 지능형 로봇