JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 21, No. 6, pp. 1143-1152, Jun. 2020

Check for updates

소프트웨어 제어 흐름 그래프의 구조적 유사성 비교를 위한 접근 방법

임 현 일
경남대학교 컴퓨터공학부 부교수

An Approach to Identifying Structural Similarity of Control Flow Graphs of Software

Hyun-il Lim
Associate Professor, Department of Computer Engineering, Kyungnam University, Gyeongsangnam-do 51767, Korea

[요 약]

제어 흐름 그래프는 소프트웨어의 특성을 나타내기 위한 기본적인 데이터 구조이다. 소프트웨어의 특성을 분석하는 것은 소프트웨어 개발 및 소프트웨어 표절 탐지 또는 악성 프로그램 탐지 등 다양한 응용 분야에 유용하게 활용되고 있다. 본 논문에서는 소프트웨어 분석 방법으로 제어 흐름 그래프의 구조적 유사성을 계산하기 위해 기본 블록의 분석과 에지의 연결 관계를 반영하는 방법을 제안한다. 제어 흐름 그래프의 구조는 기본 블록과 에지의 관계를 통해 표현되기 때문에 기본 블록의 특성과 에지의 관계를 분석하고 두 제어 흐름 그래프에서 구조적으로 유사한 기본 블록의 매칭을 찾는 방법을 적용한다. 매치된 기본 블록의 관계로 부터 두 제어 흐름 그래프 사이의 구조적 유사성을 계산한다. 본 논문의 구조적 유사성 비교 방법을 평가하기 위해 서로 다른 실행 구조를 가진 자바 프로그램을 대상으로 실험을 수행하였다. 실험 결과에서 본 논문에서 제안한 방법은 소프트웨어의 구조적 특성에 따라 구조적 유사성을 비교하는 데 효과적으로 적용될 수 있음을 확인할 수 있었다.

[Abstract]

The control flow graph is a basic structure for representing characteristics of software. Analyzing characteristics of software is useful in software development and various application areas, including software plagiarism detection or malware detection. In this paper, an analysis method is proposed to reflect the relationship between the basic block and the edge to identify the structural similarity of the control flow graph. Since the structure of the control flow graph is expressed through the relationship between the basic blocks and the edges, a method is applied to find a matching of similar basic block in two control flow graphs through analysis of the relationship between the characteristics of the basic blocks and the edges. The structural similarity between two control flow graphs is calculated from the matched basic blocks. To evaluate the proposed comparison method, experiments were carried out in Java programs with different implementation structures. In the experimental results, it was confirmed that the proposed method could be applied effectively to compare structural similarities according to the structural characteristics of software.

색인어 : 프로그램 제어 흐름 분석, 소프트웨어 분석, 소프트웨어 유사도, 구조적 유사성
Key word : Program control flow analysis, Software analysis, Software similarity, Structural similarity

http://dx.doi.org/10.9728/dcs.2020.21.6.1143

Received 27 April 2020; **Revised** 15 June 2020
Accepted 25 June 2020

***Corresponding Author; Hyun-il Lim**

Tel: +82-55-249-2650
E-mail: hilim@kyungnam.ac.kr

Ⅰ. 서론

제어 흐름 그래프는 프로그램의 정적인 구조와 동적인 동작 특성을 함께 보여줄 수 있는 기본적이고 중요한 데이터 구조다. 또한 제어 흐름 그래프를 이용해 코드 최적화, 유사 코드 검출, 표절 코드 탐색 등 다양한 분야에 활용할 수 있다. 예를 들어 프로그램 간 유사성을 식별하기 위한 기본적인 비교 방법으로 제어 흐름 그래프의 비교를 사용할 수 있다. 최근에는 악성 소프트웨어 검출을 위해 프로그램의 제어 흐름 그래프를 이용한 매칭 방법을 적용하고 한다 [1, 2, 3, 4]. 이와 같이 제어 흐름 그래프의 비교 방법은 응용 프로그램 및 소프트웨어 분석의 여러 분야에 폭넓게 적용되고 있다.

소프트웨어의 제어 흐름 그래프는 프로그램의 명령어를 포함하고 제어 흐름으로 연결되는 기본 블록을 이용하여 프로그램의 구조적 특성을 효과적으로 표현할 수 있는 데이터 구조이다. 제어 흐름 그래프는 소프트웨어의 구조적 특성을 다양한 측면에서 보여줄 수 있기 때문에 소프트웨어의 특징을 파악하기 위해 정적 또는 동적 소프트웨어 분석에 널리 이용되고 있다. 그래서 소프트웨어 표절 탐지 [5, 6]나 악성코드 탐지 등과 같이 다양한 응용 분야에서 소프트웨어를 식별하고 인식하기 위해 제어 흐름 그래프를 활용한다. 또한, 소프트웨어의 특성을 제어 흐름 그래프를 통해 효과적으로 표현함으로써 빅데이터나 기계 학습과 같은 다양한 응용 분야에서 분석 데이터로 사용될 수 있다.

본 논문에서는 소프트웨어의 제어 흐름 그래프에 대해 유사한 기본 블록과 에지를 매칭하는 방법을 이용하여 소프트웨어 간의 구조적 유사성을 식별하는 방법을 제안한다. 이 방법은 소프트웨어의 기본 블록 및 제어 흐름의 정적 구조를 분석하고 비교함으로써 구조적 유사성을 파악할 수 있다. 소프트웨어의 제어 흐름 그래프 사이에 구조적 유사성을 파악하기 위해 분석된 정보에 따라 두 제어 흐름 그래프의 기본 블록 간 유사한 기본 블록을 찾아내고, 기본 블록의 매칭으로부터 제어 흐름 그래프의 구조적 유사성을 분석한다. 이 과정에서 기본 블록의 특성과 기본 블록의 들어오는 에지와 나가는 에지가 표현하는 제어 흐름을 고려함으로써 제어 흐름 그래프가 가지는 구조적 특징을 반영한다.

제안된 방법의 성능을 평가하기 위해 서로 다른 실행 구조를 가지는 동일한 작업을 수행하는 Java 프로그램에 대한 실험을 수행한다. 실험 결과에서 제안된 비교 분석 방법은 제어 흐름 그래프의 구조를 반영할 수 있는 구조적 유사성을 분석하기 위해서 효과적인 방법임을 확인할 수 있다.

본 논문은 다음과 같이 구성된다. 제2장에서는 제어 흐름 그래프 비교에 대한 관련 연구를 요약한다. 제3장에서는 소프트웨어의 제어 흐름 그래프의 구조적 유사성을 설명하고, 구조적 유사성을 구하는 방법을 제안한다. 제4장에서는 본 연구에서 수행한 실험 환경과 실험 결과를 보여주고 제안한 방법의 성능을 개선하기 위한 개선 방향에 대해 설명한다. 마지막으로, 제5장에서 이 논문의 결론을 맺는다.

Ⅱ. 관련 연구

그래프는 다양한 종류의 데이터를 저장하고 관리하기 위해 여러 분야에서 널리 사용되는 데이터 구조이다. 그래프는 정의되는 형태에 따라 복잡한 데이터를 효율적으로 표현할 수 있다. 이런 그래프 형태로 표현되는 데이터의 특성을 비교하기 위해서 그래프 구조의 특성을 이해하고 비교하는 방법이 필요하다. 그래프로 표현된 데이터 구조를 효과적으로 비교하기 위해 그래프 구조에 포함되는 특징을 효율적으로 반영할 수 있는 비교 방법을 설계해야 한다.

그래프 노드의 레이블(label)은 데이터의 특성을 나타내는 중요한 요소이며, 그래프 비교 방법에서 노드의 레이블을 반영하는 방법에 따라 두 가지 접근 방법으로 분류할 수 있다. 그래프의 노드에 레이블이 없는 경우에는 노드는 동일한 특성을 가진 것으로 인식되기 때문에 노드를 연결하는 에지의 관계를 고려하는 방법을 이용한다. 반면, 그래프 노드에 구별 가능한 레이블이 있는 경우에 노드의 레이블은 그래프의 비교에서 유사한 노드를 찾는데 중요한 정보가 된다. 그래프는 데이터를 저장하고 관리하기 위해 널리 사용되는 데이터 구조이기 때문에 유사성 비교를 위한 연구는 활발히 진행되어 왔다. 전통적인 그래프의 비교에서 활용되는 부분 그래프의 유사성을 위한 최대 공통 에지 서브그래프 (maximum common edge subgraph) 알고리즘 [7, 8], 그래프의 편집 거리 (edit distance) [9] 등의 방법은 노드에 레이블이 구별되지 않는 일반적인 그래프 비교 방법으로 적용할 수 있다.

소프트웨어의 구조를 나타내는 제어 흐름 그래프의 구조적 특성을 비교하기 위해서는 제어 흐름 그래프가 가지는 특성을 반영할 수 있다. Nagarajan 등 [10]은 제어 흐름 그래프의 비교에서 함수의 호출 정보를 수집하고 프로그램에서 호출되는 함수의 정보를 비교하는 방법을 제안하였다. 함수의 호출 정보는 프로그램의 실행에 중요한 부분을 담당하기 때문에 함수 호출이 빈번한 프로그램의 비교에서 효과적인 결과를 얻을 수 있다. Qiu 등 [5]은 자바 프로그램의 표절 탐지를 위해 제어 흐름 그래프를 비교하는 방법을 제안하였다. 자바 바이트코드 명령어를 기준으로 기본 블록을 매칭한 후 기본 블록에 대한 경로를 비교하는 2단계의 매칭 과정을 통해 제어 흐름 그래프를 비교하였다. Kapoor 등 [2]은 악성 코드의 탐지를 위한 방법으로 제어 흐름 그래프 비교를 통해 다중 클래스 분류를 지원하는 방법을 제안하였다. 이 방법은 제어 흐름 그래프에서 특성값을 추출하고, 문서의 특성을 비교하기 위해 사용되는 분석 방법인 tf-idf 가중치 [11]를 이용하여 프로그램을 분류하였다. Alasmary 등 [3, 4]은 악성 코드 분석에 적용할 수 있는 그래프 비교 방법을 제안하였다. 이 방법은 제어 흐름 그래프가 가지는 노드와 에지의 개수, 노드 사이의 최단 경로, 부분 그래프(subgraph)의 구성 요소 등 제어 흐름 그래프의 특성 정보를 비교하였다. Dijkman 등

[12]과 Isah 등 [6]은 그래프의 구조적 유사성을 비교하기 위해서 그래프 편집 거리를 이용하는 방법을 제안하였다. 이 방법은 편집 연산을 통해서 그래프의 다른 정도를 측정하고 유사도를 구할 수 있다. 기존의 접근 방법은 악성 코드 탐지, 표절 탐지 등의 분야에서 제어 흐름 그래프가 활용되고 있으며, 그래프 구성 요소 가지는 요소적인 특성을 비교하고 있다. 이 논문에서는 제어 흐름 그래프의 구조적 특징을 반영할 수 있는 기본 블록의 비교 방법을 제안한다.

Ⅲ. 제어 흐름 그래프

3-1 제어 흐름 그래프의 개념

소프트웨어의 제어 흐름 그래프 [13, 14]는 프로그램의 정적인 구조 뿐만 아니라 동적인 특성을 나타내기 위해 소프트웨어 분석에 널리 사용된다. 일반적으로 제어 흐름 그래프는 그래프 $G = (N, E)$로 정의되며, 여기서 N과 E는 각각 제어 흐름 그래프 G의 노드 집합과 에지 집합을 나타낸다. 제어 흐름 그래프의 각 노드는 프로그램의 기본 블록을 나타내며, 프로그램 실행 중에 실행되는 구문 정보와 명령어들을 포함하고 있다. 제어 흐름 그래프의 각 에지는 기본 블록 사이에 제어 흐름을 나타내며 프로그램 실행 중 기본 블록 사이에 실행 가능한 기본 블록의 흐름을 보여준다.

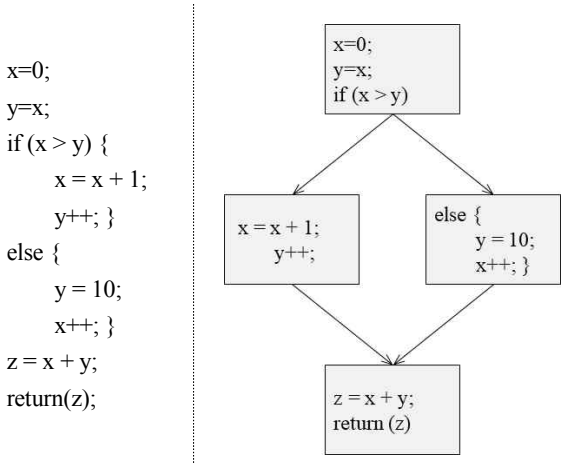


그림 1. if 분기문을 포함하는 프로그램의 제어 흐름 그래프 예제

Fig. 1. An example of the control flow graph of a program that contains an if branch

그림 1은 프로그램의 제어 흐름 그래프의 예제를 보여주고 있다. 왼쪽의 예제 프로그램은 if 분기문을 포함하고 있으며, 이 프로그램에 대한 제어 흐름 그래프는 오른쪽의 그래프에서 보여주고 있다. 이 예제에서는 if 분기문에서 실행 가능한 기본 블록의 관계가 그래프로 표현되고 있다. 이와 같이 분기문 또는 반복문의 실행 흐름을 나타내는 구조적 특성이 그래프의 에지로 표현되기 때문에 제어 흐름 그래프는 프로그램의 실행 경로 및 실행 구조를 분석하고 표현하는 데 효과적으로 활용된다.

제어 흐름 그래프는 소프트웨어의 구문 및 실행 구조의 특징을 정적으로 표현하기 위해 효과적인 데이터 구조이다. 또한 소프트웨어 실행 흐름을 통해 실행 시간에 수행 가능한 작업을 분석할 수 있다. 제어 흐름 그래프는 소프트웨어의 다양한 특성들을 표현하기 때문에 소프트웨어 분석을 위한 빅데이터 또는 기계 학습 방법 등에서 분석에 필요한 기본 데이터로 사용될 수 있다.

3-2 제어 흐름 그래프의 구조적 유사성

소프트웨어의 제어 흐름 그래프는 소프트웨어의 특성을 표현하기 위한 그래프 데이터 구조의 특별한 형태로 표현된다. 제어 흐름 그래프는 노드와 에지의 집합으로 표현되며, 노드와 에지는 각각 기본 블록과 소프트웨어의 실행 가능한 제어 흐름을 표현한다. 구조적 유사성 [6, 12]은 작업의 선후 관계를 표현하는 그래프의 구조를 통해서 프로그램에 포함된 순차, 반복, 분기 등의 구조적 특성에 대한 유사도를 의미한다. 제어 흐름 그래프의 구조적 유사성을 인식하기 위해 일반적인 그래프 비교 방법을 적용할 수 있지만 기존의 그래프 비교 방법은 소프트웨어의 제어 흐름 그래프에서 나타나는 기본 블록의 특성과 선후 관계를 반영하기에 적합하지 않다. 이를 개선하기 위해 제어 흐름 그래프의 구조적 특성을 비교할 수 있는 기본 블록의 특성과 실행 구조를 나타내는 에지의 관계를 고려하는 것이 필요하다.

제어 흐름 그래프의 특징으로서 기본 블록은 소프트웨어에 포함된 명령어와 순서를 포함하고 있으며 프로그램의 실행 구조를 나타내는 구문 정보를 포함하고 있다. 이런 기본 블록의 특성을 유사한 기본 블록을 구별하거나 인식하는 데 반영하면 유사성을 가진 블록을 찾아내는 데 효과적인 결과를 얻을 수 있다. 본 논문에서는 제어 흐름 그래프 간에 유사한 기본 블록을 인식하기 위해 제어 흐름 그래프가 가지는 특징 중에서 기본 블록의 특성과 에지의 제어 흐름 정보를 활용한다. 기존의 그래프 비교 방법에서 그래프에 포함된 노드의 레이블은 그래프 사이에 동일한 레이블이 있는 노드의 일치성을 확인하는데 적용되었다. 기존의 접근 방식에서 부분적으로 유사한 내용을 포함하는 노드를 변별력있게 구별하지 못하기 때문에 제어 흐름 그래프를 비교할 때 구조적 유사성을 반영하기 어려운 점이 있다. 반면 부분적인 노드의 유사도를 고려하고 매칭에 반영한다면 구조적 유사성을 인식하는데 도움이 될 것이다.

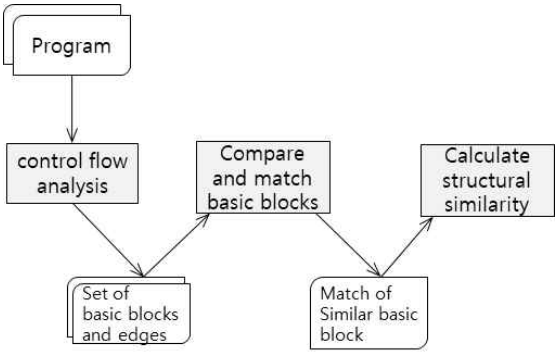


그림 2. 제어 흐름 그래프의 구조적 유사도를 분석하는 과정

Fig. 2. The procedure for analyzing structural similarity of control flow graphs

그림 2는 본 논문에서 제안한 프로그램의 구조적 유사도를 분석하는 과정을 보여주고 있다. 제어 흐름 분석을 통해 기본 블록과 에지의 정보를 분석한 후 기본 블록과 에지 특성을 비교해서 유사성을 평가하고 유사도를 계산한다. 유사도가 높은 기본 블록의 쌍은 기본 블록에 포함된 명령어와 에지의 연결 정보를 함께 반영하기 때문에 기본 블록의 연결을 통해 이루어지는 구조적 특성을 유사도에 반영할 수 있다. 기본 블록에 대한 구조적 유사도를 기준으로 두 프로그램 사이에 유사도가 높은 기본 블록의 쌍을 매칭함으로써 두 소프트웨어의 제어 흐름으로부터 기본 블록의 구조적 유사도를 찾을 수 있다. 최종적으로 매칭된 기본 블록으로 부터 프로그램의 구조적 유사성을 계산한다.

3-3 기본 블록의 특성 비교

제어 흐름 그래프의 구조적 유사성을 판단하기 위해서 우선 제어 흐름 그래프사이에 유사한 기본 블록의 쌍을 찾는 것이 필요하다. 프로그램의 기본 블록을 분석하기 위해서 각 기본 블록의 시작점이 되는 헤더를 찾고 기본 블록의 헤더로부터 기본 블록들을 구분한다. 프로그램에서 기본 블록의 헤더는 다음의 세 가지 조건중 하나를 만족하는 구문으로 찾을 수 있다.

(1) 프로그램의 첫 번째 구문
(2) 프로그램에서 분기문의 목적지가 되는 구문
(3) 분기문 다음에 위치하는 구문

이 세 가지 조건 중 하나를 만족하는 기본 블록의 헤더를 찾은 후 헤더를 시작으로 다음 헤더가 나오기 전까지 하나의 기본 블록을 구성한다. 각각의 기본 블록은 프로그램의 실행에서 분할되지 않고 반드시 연속적으로 실행되는 구문들로 구성된다.

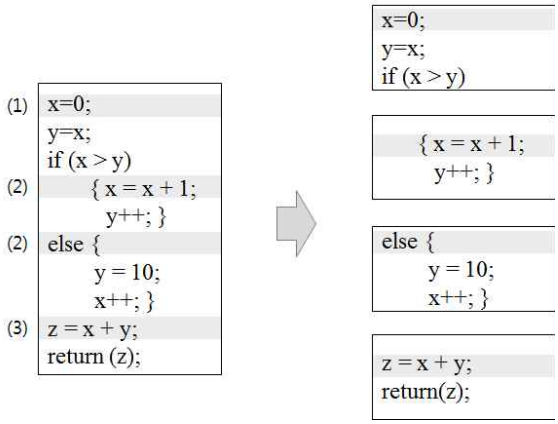


그림 3. 프로그램에서 기본 블록의 분석 예제

Fig. 3. An example of analyzing basic blocks in a program

그림 3은 그림 1에서 보여준 프로그램의 예제에 대해서 기본 블록을 분석하는 과정을 보여주고 있다. 예제 프로그램에서 기본 블록의 헤더에 해당하는 (1) 첫 번째 구문, (2) 분기문의 목적지, (3) 분기문의 다음 구문을 인식하고 이 구문을 기준으로 분할하면 그림 2의 오른쪽과 같이 네 개의 기본 블록으로 분석된다.

각 프로그램으로 부터 기본 블록을 분석한 후에 두 프로그램에서 기본 블록에 포함된 특징과 구문 정보를 비교하여 기본 블록사이에 유사도를 계산한다. 이 단계에서 기본 블록의 특징은 포함된 명령어의 집합으로 표현될 수 있으며 기본 블록 간의 유사성을 계산하기 위해 명령어 분포의 유사한 정도를 비교한다. 따라서, 기본 블록의 유사도는 기본 블록 사이에 포함된 공통된 명령어의 수와 비율을 통해서 계산한다.

예를 들어, $\{P_1, P_2, \cdots, P_m\}$와 $\{Q_1, Q_2, \cdots, Q_n\}$는 각각 두 개의 제어 흐름 그래프에서 기본 블록의 집합이라고 하자. 그러면 두 제어 흐름 그래프에서 각 i번째와 j번째 기본 블록의 유사도 $S_{Block}(P_i, Q_j)$는 다음의 수식으로 계산한다.

$$S_{Block}(P_i, Q_j) = \frac{|B(P_i) \cap B(P_j)|}{|B(P_i) \cup B(P_j)|} \quad (1)$$

여기서 $B(P_i)$는 기본 블록 P_i에 포함된 명령어의 집합을 나타내며, $|A|$은 집합 A에 포함된 원소의 수를 나타낸다.

수식 (1)에서 기본 블록의 유사성은 공통으로 포함된 명령어의 수와 비율을 계산하여 측정한다. 이 기본 블록의 유사도는 직관적으로 기본 블록 쌍들 사이에 공통된 명령어가 많을수록 높은 유사도 값을 가진다. 기본 블록 사이의 유사도 결과는 유사한 기본 블록의 쌍을 찾아내고 제어 흐름 그래프 사이에 기본 블록을 매칭하는데 사용된다.

또한 제어 흐름 정보는 제어 흐름 그래프의 구조적 특성을 구별할 수 있는 구문적 정보를 제공하기 때문에 기본 블록 사이

의 제어 흐름을 표현하는 에지 정보를 기본 블록의 매칭에 반영할 필요가 있다. 기본 블록을 비교하는 것과 유사한 방법으로 기본 블록에 대해 들어오는 에지(incoming edge)와 나가는 에지(outgoing edge) 사이의 유사성을 측정하고 기본 블록의 매칭에 반영한다. 따라서, 제어 흐름 그래프에서 유사한 기본 블록을 매칭할 때, 기본 블록과 제어 흐름 에지로 연결된 인접한 기본 블록과의 연관성을 고려하는 것이 필요하다.

소프트웨어에서 기본 블록에 대해 들어오는 실행 흐름과 나가는 실행 흐름의 관계는 제어 흐름 그래프의 전체 구조 내에서 기본 블록의 위치와 실행 상태 등을 반영하며 프로그램의 실행 과정에 필요한 구조적 특성을 반영한다. 따라서 기본 블록의 제어 흐름을 고려함으로써 유사한 기본 블록에 대한 구조적으로 비슷한 위치의 기본 블록을 매칭하기 위해 효과적인 접근법이 된다.

예를 들어, 두 개의 기본 블록 P_i와 Q_j가 있을 때, 기본 블록 P_i와 Q_j의 나가는 제어 흐름의 유사도는 다음의 수식으로 계산할 수 있다.

$$S_{out}(P_i, Q_j) = \frac{|\{B(x)|x \in OUT(P_i)\} \cap \{B(y)|y \in OUT(Q_j)\}|}{|\{B(x)|x \in OUT(P_i)\} \cup \{B(y)|y \in OUT(Q_j)\}|} \quad (2)$$

여기서 $OUT(P)$는 기본 블록 P에서 나가는 에지를 통해 도달할 수 있는 기본 블록 집합을 의미한다.

나가는 제어 흐름 에지의 유사도는 기본 블록 실행 이후에 나가는 에지를 통해 실행될 수 있는 블록으로 결정되며, 공통으로 포함된 명령어의 수와 비율에 의해 계산된다. 직관적으로, 나가는 에지의 유사도는 기본 블록이 실행된 후 실행되는 블록에 동일한 명령어를 많이 포함할수록 높아진다. 이 수치는 제어 흐름 그래프에서 각 기본 블록에 인접한 구문의 구조적 유사성을 반영할 수 있기 때문에 유사한 기본 블록을 매칭할 때 보다 안정적인 결과를 얻는데 도움이 된다.

이와 유사한 방법으로 기본 블록 P_i와 Q_j의 들어오는 제어 흐름 에지의 유사도 $S_{in}(P_i, Q_j)$는 다음과 같이 계산할 수 있다.

$$S_{in}(P_i, Q_j) = \frac{|\{B(x)|x \in IN(P_i)\} \cap \{B(y)|y \in IN(Q_j)\}|}{|\{B(x)|x \in IN(P_i)\} \cup \{B(y)|y \in IN(Q_j)\}|} \quad (3)$$

여기서 $IN(P)$는 들어오는 에지를 통해 기본 블록 P에 도달할 수 있는 기본 블록의 집합을 의미한다.

기본 블록을 비교할 때, $S_{Block}(P_i, Q_j)$는 기본 블록 내부의 유사성을 비교한 결과를 보여준다. 또한, $S_{out}(P_i, Q_j)$와 $S_{in}(P_i, Q_j)$는 기본 블록의 비교에서 기본 블록과 인접한 제어 흐름의 유사성을 나타낸다. 기본 블록을 비교할 때 위의 세 가지 요소를 함께 고려함으로써 기본 블록 내부에 포함된 명령어뿐만 아니라 제어 흐름 에지를 통한 구조적 특성과 기본 블록의 실행 순서를 비교 결과에 반영할 수 있다. 따라서 제어 흐름 그래프 사이에 유사한 기본 블록을 매칭할 때 보다 구조적으로 유사한 위치의 기본 블록을 찾는데 도움이 된다.

기본 블록에 대해서 세 가지 유사도 비교 결과를 이용해 기본 블록P_i와 Q_j의 구조적 유사도 $S(P_i, Q_j)$는 다음과 같이 계산한다.

$$S_{raw}(P_i, Q_j) = \frac{S_{Block}(P_i, Q_j) + S_{out}(P_i, Q_j) + S_{in}(P_i, Q_j)}{3}$$
$$S(P_i, Q_j) = \begin{cases} S_{raw}(P_i, Q_j) & \text{if } S_{raw}(P_i, Q_j) > \tau \\ 0 & otherwise \end{cases} \quad (4)$$

여기서 τ는 기본 블록의 유사성을 식별하기 위해 기본 블록 사이의 매칭을 위한 유사도의 임계값을 나타낸다. 이 값은 기본 블록의 세 가지 유사도 측정 결과를 함께 고려하여 제어 흐름 그래프에 포함된 기본 블록의 유사도를 결정한다. 기본 블록을 비교할 때 세 가지 유사도의 평균값이 τ보다 작을 때는 0으로 결정함으로써 유사하지 않은 기본 블록을 매칭에서 제외하도록 설계하였다. 기본 블록과 제어 흐름 정보의 특성이 모두 동일하면 완전하게 일치하기 때문에 유사도 $S(P_i, Q_j)$는 최대값 1.0이 된다. 반면, 기본 블록과 그 제어 흐름이 서로 차이를 보이는 경우 유사도는 0.0이 될 것이다. 따라서 기본 블록의 유사도는 기본 블록과 인접한 제어 흐름의 유사한 정도에 따라 0.0에서 1.0 사이에서 결정된다. 이 값은 두 개의 제어 흐름 그래프 사이에 구조적 유사성을 식별하기 위해 유사한 기본 블록을 찾아서 매칭하는 데 적용된다.

두 제어 흐름 그래프 사이의 모든 기본 블록 쌍을 비교하여 기본 블록의 유사도를 비교하고, 유사한 기본 블록의 매칭 결과를 찾는다. 매칭된 기본 블록으로부터 제어 흐름 그래프의 구조적 유사성을 인식할 수 있다. 즉, 각각 m개와 n개의 기본 블록이 있는 2개의 제어 흐름 그래프의 경우, 제어 흐름 그래프 사이의 $m \times n$ 쌍의 기본 블록에 대한 유사도를 계산한다. 기본 블록의 유사도로부터 유사도가 높은 기본 블록의 매칭 결과를 찾고, 매칭된 기본 블록의 유사도를 이용해 전체 제어 흐름 그래프의 구조적 유사성을 식별한다.

그림 4는 두 개의 다른 제어 흐름 그래프에 대해 기본 블록 P와 Q의 예를 보여준다. 두 개의 기본 블록은 다른 기본 블록과 에지로 연결되어 있으며, 들어오는 에지와 나가는 에지를 통해 기본 블록의 구조적 특성을 표현한다. 이 예제에서 기본 블록에 있는 집합은 기본 블록에 포함된 명령어 집합을 나타낸다.

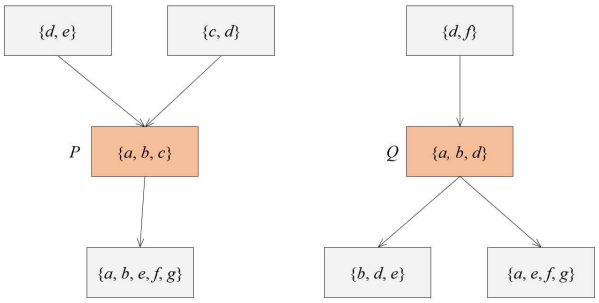


그림 4. 제어 흐름 그래프에서 기본 블록 *P*와 *Q*의 예제
Fig. 4. An example of basic blocks *P* and *Q* in control flow graphs

이 예제에서 기본 블록 P와 Q는 각각 세 개의 명령어 $\{a,b,c\}$와 $\{a,b,d\}$를 가지고 있다. 따라서 두 기본 블록 사이의 유사도 $S_{Block}(P,Q)$는 다음과 같이 계산된다.

$$S_{Block}(P,Q)=\frac{|\{a,b,c\}\cap\{a,b,d\}|}{|\{a,b,c\}\cup\{a,b,d\}|}=\frac{|\{a,b\}|}{|\{a,b,c,d\}|}=\frac{1}{2}$$

그리고, 기본 블록의 나가는 에지와 들어오는 에지의 유사도 $S_{out}(P_i,Q_j)$와 $S_{in}(P_i,Q_j)$는 각각 다음과 같이 계산된다.

$$S_{out}(P,Q)=\frac{|\{a,b,e,f,g\}\cap\{a,b,d,e,f,g\}|}{|\{a,b,e,f,g\}\cup\{a,b,d,e,f,g\}|}=\frac{|\{a,b,e,f,g\}|}{|\{a,b,d,e,f,g\}|}=\frac{5}{6}$$

$$S_{in}(P,Q)=\frac{|\{c,d,e\}\cap\{d,f\}|}{|\{c,d,e\}\cup\{d,f\}|}=\frac{|\{d\}|}{|\{c,d,e,f\}|}=\frac{1}{4}$$

세 가지 유사도 비교 결과에서 $\tau=0.3$ 일 때, 두 기본 블록 P와 Q의 유사도 $S(P,Q)$는 다음과 같다.

$$S(P,Q)=\frac{S_{Block}(P_i,Q_j)+S_{out}(P_i,Q_j)+S_{in}(P_i,Q_j)}{3}=\frac{\frac{1}{2}+\frac{5}{6}+\frac{1}{4}}{3}=0.527$$

따라서 그림 4에 보여주는 기본 블록 P와 Q의 구조적 유사도는 0.527이다. 이 예제와 같이 두 개의 제어 흐름 그래프 사이의 모든 기본 블록 쌍의 유사도는 제어 흐름 그래프 사이에 유사한 기본 블록을 매칭하기 위해 필요하다.

3-4 매칭을 통한 구조적 유사성

두 개의 제어 흐름 그래프 A와 B에서 m과 n은 각 제어 흐름 그래프에 포함된 기본 블록의 개수라고 할 때, 두 제어 흐름 그래프 사이에 $m\times n$ 개의 기본 블록 쌍에 대한 유사도를 구한다. 이 기본 블록의 쌍 중에서 유사도가 높은 순서대로 기본 블록의 쌍을 매칭하도록 시도한다. 두 제어 흐름 그래프에 포함된 기본 블록의 수가 다른 경우에는 작은 그래프에 포함된 모든 기본 블록이 매칭될 수 있도록 매칭이 이루어지게 된다. 따라서, 두 제어 흐름 그래프 A와 B의 매칭 결과 $M(A,B)$는 제어 흐름 그래프 사이에서 $\min(m,n)$개의 가장 유사한 기본 블록의 쌍으로 구성된다. 전체 제어 흐름 그래프의 구조적 유사도는 매칭된 기본 블록의 유사도를 모두 반영한 평균값으로 계산할 수 있다. 따라서, 제어 흐름 그래프 A와 B사이의 구조적 유사도 $Sim(A,B)$는 다음과 같이 계산한다.

$$Sim(A,B)=\frac{\sum_{(P_i,Q_j)\in M(A,B)}S(P_i,Q_j)}{\min(m,n)} \quad (5)$$

여기서 $M(A,B)$는 제어 흐름 그래프 A와 B 사이에 매칭된 기본 블록의 쌍을 의미한다. 수식 (5)에서 전체 그래프의 구조적 유사도를 계산하기 위해 매치가 이루어진 기본 블록 쌍의 유사도를 모두 합하고, 매칭된 갯수로 나누어서 구조적 유사도를 구할 수 있다. 따라서 결과 값은 제어 흐름 그래프사이에 매치가 이루어진 기본 블록의 구조적 유사도 정도에 따라 0과 1 사이에서 정해진다.

유사도를 구하는 과정에서 $M(A,B)$로 표현된 기본 블록의 매칭은 기본 블록과 제어 흐름의 구조적 특징이 유사한 블록을 매칭하고, 두 제어 흐름 그래프 사이에 유사한 기본 블록에 대해 우선 매칭이 이루어진다. 따라서 제어 흐름 그래프에 유사한 구조가 많으면 높은 유사도를 가진 기본 블록들이 더 많이 매칭될 수 있다. 따라서 기본 블록의 구조적 유사도는 $M(A,B)$를 통해서 제어 흐름 그래프의 구조적 특징을 효과적으로 반영하게 된다.

예를 들어, 두 개의 제어 흐름 그래프 A와 B에 대해 $\{P_1,P_2,P_3,P_4\}$과 $\{Q_1,Q_2,Q_3,Q_4\}$는 각각 기본 블록의 집합이라고 하자. 제어 흐름 그래프 간의 기본 블록 쌍의 구조적 유사도는 수식 (4)에 의해 계산된다. 표 1은 두 제어 흐름 그래프 사이에 기본 블록의 구조적 유사도 계산 결과의 예를 보여준다.

표 1. 제어 흐름 그래프 사이의 기본 블록의 구조적 유사도 예제
Table 1. An example of structural similarities of basic blocks between control flow graphs.

	P_1	P_2	P_3	P_4
Q_1	0.7	0.2	0.4	0.4
Q_2	0.0	0.9	0.7	0.5
Q_3	0.4	0.5	0.5	0.8
Q_4	0.5	0.7	0.6	0.5

이 예에서 두 제어 흐름 그래프로 부터 유사한 기본 블록을 매칭하고 구조적 유사성을 식별하는 과정은 다음과 같다. 우선 유사한 기본 블록을 매칭하기 위해서 유사도가 높은 기본 블록의 쌍부터 순서대로 일대일 매칭한다. 예를 들어 표 1에서 기본 블록 쌍 (P_2, Q_2)는 기본 블록의 유사도가 0.9로 가장 높다. 따라서, 이 기본 블록의 쌍이 우선 매칭된다. 남아있는 기본 블록 쌍 중에서 유사도가 높은 순서대로 기본 블록에 대한 매칭을 시도 한다. (P_4, Q_3)와 (P_1, Q_1)는 각각 기본 블록의 구조적 유사도가 0.8과 0.7이며 순서대로 매칭된다. 마지막으로 남아 있는 기본 블록 P_3 와 Q_4 가 매칭되고 이 두 기본 블록의 유사도는 0.5이다. 따라서 두 제어 흐름 그래프 A와 B사이의 기본 블록의 매칭 집합 $M(A, B)$는 { (P_2, Q_2), (P_4, Q_3), (P_1, Q_1), (P_3, Q_4) }임을 알 수 있다. 기본 블록 쌍의 매칭 집합으로 부터 제어 흐름 그래프 사이의 구조적 유사도는 다음과 같이 계산된다.

$$Sim(A,B) = \frac{\sum_{(P_i, Q_j)\in M(A,B)} S(P_i, Q_j)}{\min(m,n)} = \frac{0.9+0.8+0.7+0.5}{\min(4,4)} = \frac{2.9}{4} = 0.725$$

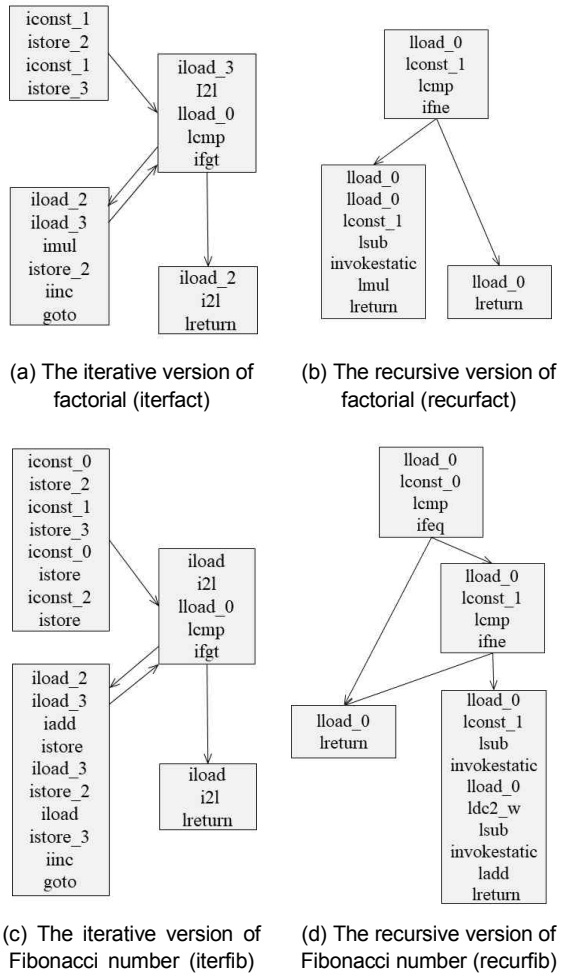
따라서 두 제어 흐름 그래프 사이의 구조적 유사도는 0.725로 측정된다. 이와 같이 높은 구조적 유사도를 보이는 결과는 두 개의 제어 흐름 그래프가 구조적으로 높은 유사성을 공유하고 있다는 것을 나타낼 수 있다.

Ⅳ. 실험 및 평가

4-1 실험 환경 구성 및 결과

본 논문에서는 제어 흐름 그래프의 구조적 유사성을 확인하기 위해 유사한 기본 블록의 쌍을 인식하고 매칭한 결과를 통해 유사도를 측정하는 방법을 제안하였다. 본 논문에서 제안한 방법을 적용하여 소프트웨어의 제어 흐름 그래프를 이용한 구조적 유사성을 비교할 수 있다.

본 논문에서 제안한 방법의 성능을 평가하기 위해 제안된 방법을 구현하고 실험을 수행하였다. 제안된 방법은 Python 언어[15]를 이용하여 구현하였으며, Java 프로그램으로부터 제어 흐름 그래프를 생성하여 실험을 수행하였다. 본 실험에서 사용한 자바 프로그램은 서로 다른 실행 구조를 포함하고 있는 프로그램으로 각각 팩토리얼과 피보나치 수를 계산하기 위한 반복 및 재귀 버전의 프로그램을 이용하였다. 이 실험에서 자바 프로그램으로 부터 컴파일된 클래스 파일에서 자바 바이트코드를 추출하였으며, 각 클래스 파일로부터 추출된 자바 바이트코드에 대해서 제어 흐름 그래프를 생성하였다. 생성된 제어 흐름 그래프에 대해 구조적 유사성을 식별하기 위해 본 논문에서 제안된 방법을 이용해 실험을 수행하였다.



(a) The iterative version of factorial (iterfact)

(b) The recursive version of factorial (recurfact)

(c) The iterative version of Fibonacci number (iterfib)

(d) The recursive version of Fibonacci number (recurfib)

그림 5. 팩토리얼과 피보나치 수를 구하는 프로그램의 제어 흐름 그래프

Fig. 5. The control flow graphs of factorial and Fibonacci number programs

그림 5는 각각 팩토리얼과 피보나치 수를 계산하는 자바 프로그램의 서로 다른 두 프로그램 버전의 제어 흐름 그래프를 보여준다. 각 제어 흐름 그래프의 기본 블록은 자바 프로그램의 실행을 위한 바이트코드 명령어들을 포함하고 있다. 반복 프로그램 버전 (a)와 (c)에서는 프로그램의 반복 구조를 수행하기 위해서 이전의 기본 블록으로 이동하기 위한 역방향 에지(back edge)가 있음을 확인할 수 있다. 반면, 재귀 구조를 가지는 프로그램 버전 (b)과 (d)에서는 프로그램들이 팩토리얼과 피보나치 수를 계산하기 위한 재귀 호출을 가지지만 제어 흐름 그래프에서 역방향 에지는 나타나지 않는다.

표 2는 실험을 통해 제어 흐름 그래프 사이의 구조적 유사도 비교 결과를 보여준다. 구조적 유사도는 기본 블록들이 포함하는 명령어와 에지로 연결된 구조적 특징을 통해 기본 블록의 집합을 매칭하고, 매칭된 기본 블록 유사도를 반영하여 전체 프로그램의 구조적 유사도를 계산하였다. 실험 결과에서 동일한 제

어 흐름 그래프를 비교할 때, 각 기본 블록은 구조적으로 동일한 기본 블록과 완전하게 매치를 이루기 때문에 동일한 프로그램의 구조적 유사도는 모두 1.0이 됨을 확인할 수 있다.

표 2. 제어 흐름 그래프 사이의 구조적 유사성 비교 결과
Table 2. The results of structural similarities between the control flow graphs

	iterfact	recurfact	iterfib	recurfib
iterfact	1.00	0.13	0.53	0.10
recurfact		1.00	0.13	0.76
iterfib			1.00	0.10
recurfib				1.00

서로 다른 제어 흐름 그래프를 비교하는 경우 프로그램의 실행 구조에 따라 유사한 실행 구조를 가진 제어 흐름 그래프의 구조적 유사도는 그렇지 않은 그래프보다 높은 유사도를 나타내는 것을 확인할 수 있다. 이 실험에서 사용한 반복 버전과 재귀 버전의 프로그램을 비교할 때, 재귀 버전의 팩토리얼(recurfact)와 피보나치 수 (recurfib)를 구하는 프로그램 사이의 구조적 유사도는 0.76이었다. 재귀 버전의 프로그램 구조는 다른 반복 구조보다 구조적으로 유사한 특성을 가진다는 것을 보여준다. 반복 버전의 팩토리얼 (iterfact)와 피보나치 수 (iterfib) 사이의 구조적 유사도는 0.53이었고, 이 값은 유사성 판단의 기준이 되는 50%보다 높은 구조적 유사도를 보여주고 있다. 반면 실행 구조가 다른 제어 흐름 그래프를 비교할 때 구조적 유사도는 50% 미만이었다. 팩토리얼 계산을 위한 반복 버전(iterfact)과 재귀 버전(recurfact)의 구조적 유사도는 0.13이었다. 또한, 피보나치 수를 구하는 반복 버전 (iterfib)과 재귀 버전 (recurfib) 사이의 구조적 유사도는 0.10으로 제어 흐름 그래프의 모든 비교 중 가장 낮은 구조적 유사도를 보여주고 있다.

실험 결과로부터 유사한 실행 구조를 가진 제어 흐름 그래프의 비교는 높은 유사도 결과를 보여주지만, 실행 구조가 다른 제어 흐름 그래프의 비교에서는 동일한 결과를 계산하는 프로그램이더라도 구조적 유사도는 변별력있게 낮은 결과를 보여주었다. 실험 결과로부터 본 논문에서 제안한 방법은 제어 흐름 그래프 사이의 구조적 특성을 반영하고, 구조적 유사도를 식별할 수 있는 방법임을 확인할 수 있다.

4-2 기존 연구와의 비교 및 개선 방향

프로그램의 유사성을 확인하기 위해서 제어 흐름 그래프를 비교하기 위한 다양한 연구가 있다. 최근에는 악성 코드의 탐지 및 표절 탐지를 위해 소프트웨어의 유사성을 비교하는 방법에도 적용되고 있다. Nagarajan 등 [10]과 Qiu 등 [5]은 제어 흐름 그래프의 비교에서 함수의 호출 정보 및 자바 바이트코드 명령어를 비교하여 매칭하였다. 그리고, Kapoor 등 [2]과 Alasmary 등 [3, 4]은 악성 코드의 탐지를 위한 방법으로 제어 흐름 그래프의 특성 값을 비교하였다. 이 방법들은 제어 흐름 그래프가 가지는 구성 요소의 특징을 비교하기 때문에 특징적인 구성 요소 (함수 호출, 바이트코드, 노드와 에지의 정보)에서 나타나는 프로그램의 특징을 비교하는 데 효과적이다. 이 방법은 제어 흐름 그래프에서 나타나는 구조적인 특징을 반영하기 보다는 구성 요소 사이의 특징을 비교하고 있다. Dijkman 등 [12]과 Isah 등 [6]은 그래프의 구조적 유사성을 비교하기 위해서 그래프 편집 거리를 이용하였으며, 편집 연산을 이용한 구조적 유사도를 구하는 방법을 이용하였다. 그래프 편집 거리는 두 그래프의 차이를 비교하기 위해서 다른 그래프를 편집 연산을 통해서 매칭시키는 방법이다. 따라서, 동일한 구성 요소를 가지고 있는 유사한 그래프에서 편집 오차를 통해 구조적 유사도를 구할 수 있다. 편집 거리를 이용하는 방법은 제어 흐름 그래프에 나타나는 일반적인 차이점 뿐만 아니라 구조적 특징도 함께 편집 거리를 측정하게 되는데, 구조적 특징에 대한 유사성과 독립적인 편집 오차를 측정할 수 있기 때문에 구조적 유사성을 반영하는데 제약이 따른다. 또한, 서로 상이한 그래프를 비교하는 경우에는 프로그램에서 유사 실행 구조를 가지더라도 편집 거리가 증가하게 되면서 구조적 유사도를 반영하는데 어려움이 있다. 상대적으로 본 논문에서 제안한 방법은 서로 상이한 그래프를 비교하는 경우에도 기본 블록을 매칭하는 과정에서 블록과 연결된 에지의 연관성을 함께 반영하기 때문에 구조적 특징을 반영하는데 유리하다.

향후 다음과 같은 추가 연구를 통해 본 논문의 구조적 유사도 비교 결과에 대한 정확도를 개선하고자 한다. 제어 흐름 그래프 사이에 유사한 기본 블록을 매칭할 때 기본 블록의 특징뿐만 아니라 들어오는 에지 및 나가는 에지를 함께 고려하고 있다. 기본 블록의 매칭을 결정할 때는 세 가지 유사도 요소의 적절한 균형을 유지함으로써 제어 흐름 그래프의 구조적 유사성을 효과적으로 표현할 수 있다. 본 논문에서 제안한 비교 방법에서 기본 블록과 나가는 에지 및 들어오는 에지의 유사도 적용 비율을 동일하게 적용하였다. 각각의 기본 블록은 일반적으로 두 개 이상의 제어 흐름 에지를 가지기 때문에 제어 흐름 그래프의 구조적 특성을 반영하기 위해 에지의 유사도를 적극적으로 반영할 필요가 있다. 따라서 소프트웨어가 가지는 특성에 따른 기본 블록과 제어 흐름 에지의 유사도 적용 비율을 최적화하고, 구조적 유사도 결과의 신뢰도를 개선할 수 있는 접근 방법에 대해서 연구하고자 한다. 또한, 구조적 유사성을 비교하기 위한 기존의 방법들과의 비교 실험을 통해 각 접근 방법의 정확도를 개선하기 위한 방안을 연구하고자 한다.

Ⅴ. 결론

소프트웨어의 제어 흐름 그래프는 정적 또는 실행 시간에 나타날 수 있는 동적 특징을 나타낼 수 있으며, 소프트웨어의 구조적 특징을 효과적으로 표현할 수 있기 때문에 소프트웨어 분

석에 널리 사용되고 있다. 본 논문에서는 소프트웨어의 제어 흐름 그래프의 구조적 유사도를 계산하기 위해 기본 블록과 에지의 연결 관계를 비교하는 방법을 제안하였다. 이 방법은 기본 블록 사이에 연결된 제어 흐름 에지의 특성을 고려하여 제어 흐름 그래프의 실행 구조가 유사한 기본 블록을 매칭한다. 매치된 기본 블록의 유사성을 반영하여 제어 흐름 그래프의 구조적 유사도를 계산하였다.

구조적 유사도 결과를 평가하기 위해 반복 및 재귀의 다른 실행 구조를 가지는 Java 프로그램으로 작성된 프로그램을 이용하여 실험을 수행하였다. 실험 결과에서 동일한 실행 구조를 가진 제어 흐름 그래프의 구조적 유사도는 다른 실행 구조를 가진 그래프보다 더 높은 유사성을 보였다. 실험 결과로부터 기본 블록과 에지의 매칭을 이용한 비교 방법은 제어 흐름 그래프의 실행 구조에 따라 달라지는 구조적 유사도를 확인하는데 효과적임을 확인할 수 있다. 제어 흐름 그래프는 코드 최적화, 유사 코드 및 표절 코드 탐색, 악성코드 탐지 등 다양한 분야에서 활용되고 있다. 소프트웨어의 구조적 유사성 비교는 유사한 프로그램을 인식하기 위한 데이터 분석 어플리케이션 및 소프트웨어의 특성을 분석하고 비교하는 응용 기술로 활용될 수 있을 것이다.

감사의 글

이 논문은 2017년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. NRF-2017R1D1A1B03034769).

References

[1] Danilo Bruschi, Lorenzo Martignoni, and Mattia Monga, “Detecting Self-Mutating Malware Using Control-Flow Graph Matching,” In *Proceedings of Detection of Intrusions and Malware & Vulnerability Assessment* (DIMVA 2006), LNCS 4064, pp. 129-143, 2006.

[2] Akshay Kapoor, Sunita Dhavale, “Control Flow Graph Based Multiclass Malware Detection Using Bi-normal Separation,” in *Defence Science Journal*, Vol. 66. No. 2, pp. 138-145, 2016.

[3] Hisham Alasmary, Aminollah Khormali, Afsah Anwar, Jeman Park, Jinchun Choi, Daehun Nyang, and Aziz Mohaisen, “Analyzing, Comparing, and Detecting Emerging Malware: A Graph-based Approach,” in *Proceedings of NDSS Symposium*, San Diego, California, February 2019.

[4] Hisham Alasmary, Aminollah Khormali, Afsah Anwar, Jeman Park, Jinchun Choi, Ahmed Abusnaina, Amro Awad, Daehun Nyang, and Aziz Mohaisen, “Analyzing and Detecting Emerging Internet of Things Malware: A Graph-Based Approach,” in *IEEE Internet of Things Journal*, Vol. 6, No. 5, pp. 8977-8988, Oct. 2019.

[5] Dehong Qiu, Jialin Sun, and Hao Li, “Improving Similarity Measure for Java Programs Based on Optimal Matching of Control Flow Graphs,” in *International Journal of Software Engineering and Knowledge Engineering*, Vol. 25, pp. 1171-1197, 2015.

[6] Adamu Isah, Adekunle Isiaka Obasa, and Mukhtar Hussaini, “Impact of Algorithm Plagiarism Detection using Structural, Block and Sentence Similarities of Control Flow Graphs,” in *Journal of Vocational and Technical Education*, Vol. 6 No. 1, November 2015.

[7] John W. Raymond, Eleanor J. Gardiner, and Peter Willett, “RASCAL: Calculation of Graph Similarity using Maximum Common Edge Subgraphs,” *The Computer Journal*, Vol. 45, No. 6, April 2002.

[8] Faisal N. Abu-Khzam, Nagiza F. Samatova, Mohamad A. Rizk, and Michael A. Langston, “The Maximum Common Subgraph Problem: Faster Solutions via Vertex Cover,” In *IEEE/ACS International Conference on Computer Systems and Applications* (AICCSA 2007), pp. 367-373, May 2007.

[9] Zhiping Zeng, Anthony K. H. Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou, “Comparing Stars: On Approximating Graph Edit Distance,” in *Preceedings of International Conference on Very Large Data Bases* (VLDB 2009), 2009.

[10] Vijay Nagarajan, Rajiv Gupta, Xiangyu Zhang, Matias Madou, Bjorn De Sutter, “Matching Control Flow of Program Versions,” in Proceedings of *2007 IEEE International Conference on Software Maintenance*, Oct. 2007.

[11] Bijoyan Das and Sarit Chakraborty, “An Improved Text Sentiment Classification Model Using TF-IDF and Next Word Negation,” Computing Research Repository(CoRR), 2018.

[12] R. Dijkman, M. Dumas, B. van Dongen, R. Käärik, and J. Mendling, “Similarity of Business Process Models: Metrics and Evaluation,” in *Information Systems*, Vol. 36, No. 2, pp. 498-516, 2011.

[13] Renhard Wilhelm and Dieter Maurer, *Compiler Design*, Addison-Wesley, 1995.

[14] Patrick P.F. Chan and Christian Collberg, “A Method to Evaluate CFG Comparison Algorithms,” in *Proceedings of 2014 14th International Conference on Quality Software*, Dallas, TX, pp. 95-104, 2014.

[15] Python Programming Language [internet], Avaliable: http://www.python.org/.

임현일(Hyun-il Lim)

1995년 : KAIST 전산학과 (공학사)
1997년 : KAIST 전산학과 (공학석사)
2009년 : KAIST 전산학과 (공학박사)

2009년～2010년: KAIST 전산학과 연구원
2010년～현 재: 경남대학교 컴퓨터공학부 부교수
※관심분야 : 소프트웨어 분석, 소프트웨어 보안, 인공 지능, 기계 학습, 소프트웨어 공학, 프로그래밍 언어 등