(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2017년09월21일
(11) 등록번호 10-1780233
(24) 등록일자 2017년09월14일

(51) 국제특허분류(Int. Cl.)
G06F 9/44 (2006.01) *G06F 11/28* (2006.01)
G06F 11/36 (2006.01)
(52) CPC특허분류
G06F 8/751 (2013.01)
G06F 11/28 (2013.01)
(21) 출원번호 10-2016-0050845
(22) 출원일자 2016년04월26일
심사청구일자 2016년04월26일
(56) 선행기술조사문헌
KR101568224 B1*
Kodhai, Egambaram, and Selvadurai Kanmani. "Method-level code clone detection through LWH (Light Weight Hybrid) approach." Journal of Software Engineering Research and Development, 2014.*
*는 심사관에 의하여 인용된 문헌

(73) 특허권자
고려대학교 산학협력단

(72) 발명자
이희조

김슬배

(74) 대리인
특허법인엠에이피에스

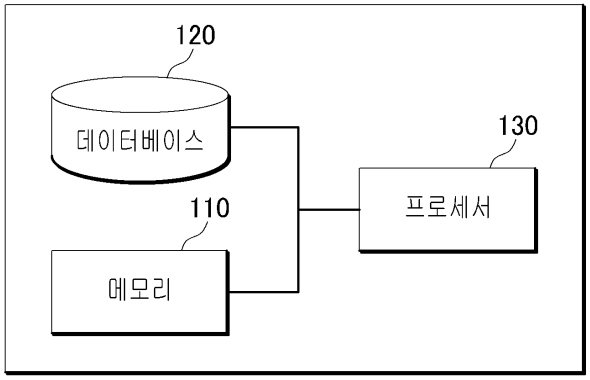
전체 청구항 수 : 총 10 항 심사관 : 서광훈

(54) 발명의 명칭 **소프트웨어의 코드 클론 탐지 장치 및 방법**

(57) 요 약

본 발명은 소프트웨어에서 코드 클론을 탐지하는 프로그램이 저장된 메모리 및 프로그램을 실행하는 프로세서를 포함한다. 이때, 프로세서는 소프트웨어에 대응하는 소스 코드로부터 복수의 함수를 추출하여 정규화 및 추상화를 수행하며, 정규화 및 추상화된 복수의 함수를 취약 코드 클론 집합과 비교하여, 소프트웨어에 대한 코드 클론 여부를 판단하되, 취약 코드 클론 집합은 복수의 타 소프트웨어에 포함된 함수로부터 추출된 후 정규화 및 추상화되어 저장된 복수의 취약 코드를 포함한다.

대 표 도 - 도1



100

(52) CPC특허분류
G06F 11/3604 (2013.01)
G06F 8/36 (2013.01)

이 발명을 지원한 국가연구개발사업
과제고유번호 1711028341
부처명 미래창조과학부
연구관리전문기관 정보통신기술진흥센터
연구사업명 SW컴퓨팅산업원천기술개발
연구과제명 IoT 소프트웨어 보안 취약점 자동 분석 기술 개발
기 여 율 1/1
주관기관 고려대학교 산학협력단
연구기간 2015.06.01 ~ 2016.05.31

명 세 서

청구범위

청구항 1

소프트웨어의 코드 클론 탐지 장치에 있어서,

소프트웨어에서 코드 클론을 탐지하는 프로그램이 저장된 메모리 및

상기 프로그램을 실행하는 프로세서를 포함하고,

상기 프로세서는 상기 소프트웨어에 대응하는 소스 코드로부터 복수의 함수를 추출하여 정규화 및 추상화를 수행하고, 정규화 및 추상화된 함수의 문자열 길이를 기초로 키를 설정하고 상기 정규화 및 추상화된 함수에 대응되는 값을 설정하여 딕셔너리를 생성하며,

상기 생성된 딕셔너리를 취약 코드 클론 집합과 비교하여, 상기 소프트웨어에 대한 코드 클론 여부를 판단하되,

상기 취약 코드 클론 집합은

딕셔너리 자료구조를 기초로, 복수의 타 소프트웨어에 포함된 함수로부터 추출된 후 정규화 및 추상화되어 저장된 복수의 취약 코드를 포함하는, 코드 클론 탐지 장치.

청구항 2

삭제

청구항 3

삭제

청구항 4

제 1 항에 있어서,

상기 프로세서는 상기 딕셔너리에 포함된 키 및 상기 취약 코드 클론 집합의 키를 비교하고,

상기 딕셔너리에 포함된 키 및 상기 취약 코드 클론 집합의 키가 일치하는 경우, 상기 딕셔너리에 포함된 키에 대응하는 값 및 상기 취약 코드 클론 집합의 키에 대응하는 취약 코드와의 비교를 수행하여 상기 소프트웨어에 대한 코드 클론 여부를 판단하는 것인, 코드 클론 탐지 장치.

청구항 5

제 1 항에 있어서,

상기 프로세서는 MD5(message digest 5) 알고리즘에 기초하여, 상기 정규화 및 추상화된 함수에 대응하는 값을 산출하는, 코드 클론 탐지 장치.

청구항 6

제 1 항에 있어서,

상기 프로세서는 상기 정규화 및 추상화된 복수의 함수를 상기 취약 코드 클론 집합에 추가하는, 코드 클론 탐지 장치.

청구항 7

제 1 항에 있어서,

상기 프로세서는 상기 추출된 복수의 함수에 포함된 화이트스페이스(whitespace) 문자를 제거하고, 상기 화이트스페이스 문자가 제거된 함수에 포함된 문자를 소문자로 변환하되,

상기 화이트스페이스 문자는 공백(space), 탭(tab) 문자 및 개행(new line) 문자를 포함하는, 코드 클론 탐지 장치.

청구항 8

제 1 항에 있어서,

상기 프로세서는 상기 추출된 복수의 함수에 포함된 변수의 자료형, 변수의 이름 및 함수 인자 각각에 대하여 미리 정해진 식별자로 변환하는, 코드 클론 탐지 장치.

청구항 9

제 1 항에 있어서,

상기 프로세서는 상기 소프트웨어에 대응하는 프로그래밍 언어에 기초하여, 정규식(regular expression)을 정의하고,

상기 정의된 정규식(regular expression)에 기초하여, 상기 함수를 추출하고, 상기 추출된 함수에 대한 정규화 및 추상화를 수행하는 것인, 코드 클론 탐지 장치.

청구항 10

코드 클론 탐지 장치에서의 소프트웨어의 코드 클론 탐지 방법에 있어서,

상기 소프트웨어에 대응하는 소스 코드로부터 복수의 함수를 추출하는 단계;

상기 추출된 복수의 함수에 대한 정규화 및 추상화를 수행하는 단계;

정규화 및 추상화된 함수의 문자열 길이를 기초로 키를 설정하고, 상기 정규화 및 추상화된 함수에 대응되는 값을 설정하여 딕셔너리를 생성하는 단계; 및

상기 생성된 딕셔너리를 취약 코드 클론 집합과 비교하여, 상기 소프트웨어에 대한 코드 클론 여부를 판단하는 단계를 포함하되,

상기 취약 코드 클론 집합은

딕셔너리 자료구조를 기초로, 복수의 타 소프트웨어에 포함된 함수로부터 추출된 후 정규화 및 추상화되어 저장된 복수의 취약 코드를 포함하는, 코드 클론 탐지 방법.

청구항 11

삭제

청구항 12

삭제

청구항 13

제 10 항에 있어서,

상기 소프트웨어에 대한 코드 클론 여부를 판단하는 단계는,

상기 딕셔너리에 포함된 키 및 상기 취약 코드 클론 집합의 키를 비교하는 단계;

상기 딕셔너리에 포함된 키 및 상기 취약 코드 클론 집합의 키가 일치하는 경우, 상기 딕셔너리에 포함된 키에 대응하는 값 및 상기 취약 코드 클론 집합의 키에 대응하는 취약 코드와의 비교를 수행하여, 상기 소프트웨어에 대한 코드 클론 여부를 판단하는 단계를 포함하는, 코드 클론 탐지 방법.

청구항 14

제 10 항 및 제 13 항 중 어느 한 항에 기재된 방법을 컴퓨터 상에서 수행하기 위한 프로그램을 기록한 컴퓨터 판독 가능한 기록 매체.

발명의 설명

기 술 분 야

[0001] 본 발명은 소프트웨어의 코드 클론 탐지 장치 및 방법에 관한 것이다.

배 경 기 술

[0002] 최근 오픈 소스 소프트웨어(open source software; OSS) 프로그램이 증가함에 따라, 개발자가 소프트웨어 개발 시 필요한 기능을 일일이 구현하지 않고, 잘 알려진 오픈 소스 소프트웨어에 구현되어 있는 코드의 일부 또는 전부를 복제하거나 재사용하는 코드 클로닝(code cloning)도 증가하고 있다. 이러한 코드 클로닝은 개발 시간 및 비용을 단축시킬 수 있다는 장점이 존재한다. 그러므로 최근 많은 개발자가 소프트웨어 개발 시 알려진 오픈 소스 소프트웨어의 코드 클로닝 사용하고 있다.

[0003] 그러나 코드 클로닝은 라이선스(license) 정책을 따르지 않은 코드 복제로 인한 오픈 소스 라이선스 위배 문제가 발생할 수 있다. 또한, 원본 소프트웨어에 버그 또는 보안 취약점이 존재하는 경우, 코드 클로닝은 버그 또는 보안 취약점 역시 복제될 수 있다. 그러므로 코드 클로닝된 소프트웨어에는 원본 소프트웨어의 버그 또는 보안 취약점이 동일하게 존재할 수 있다.

[0004] 종래의 코드 클로닝 탐지 방법은 원본 소스 코드 및 코드 클론 의심 소스 코드의 전부 또는 일부분에 포함된 소스 코드를 비교하거나, 구문 분석(parsing)에 기초하여 추출된 주요 구문 또는 토큰(token)을 비교하여 코드 클로닝을 감지할 수 있다. 그러나 이러한 종래의 방법은 문자열 비교를 수행하므로, 많은 컴퓨팅 자원을 사용하며, 코드 클론을 탐지하는데 시간이 오래 걸릴 수 있다.

[0005] 이와 관련되어, 한국 공개특허공보 제10-2014-0001951호(발명의 명칭: " 코드 클론 검출을 이용하는 지능형 코드 디퍼런싱을 수행하는 방법 및 시스템")는 대형의 복잡한 소스 코드 변경을 식별하고, 코드 디퍼런싱 툴을 이용하여, 변경을 추적하는 기술을 개시하고 있다.

발명의 내용

해결하려는 과제

[0006] 본 발명은 전술한 종래 기술의 문제점을 해결하기 위한 것으로서, 딕셔너리 자료 구조에 기초하여, 코드 클론을 탐지하는 검색 범위를 줄여 빠르고 효율적으로 소프트웨어의 코드 클론을 탐지할 수 있는 코드 클론 탐지 장치 및 방법을 제공한다.

[0007] 다만, 본 실시예가 이루고자 하는 기술적 과제는 상기된 바와 같은 기술적 과제로 한정되지 않으며, 또 다른 기술적 과제들이 존재할 수 있다.

과제의 해결 수단

[0008] 상술한 기술적 과제를 달성하기 위한 기술적 수단으로서, 본 발명의 제 1 측면에 따른 소프트웨어의 코드 클론 탐지 장치는 소프트웨어에서 코드 클론을 탐지하는 프로그램이 저장된 메모리 및 프로그램을 실행하는 프로세서를 포함한다. 이때, 프로세서는 소프트웨어에 대응하는 소스 코드로부터 복수의 함수를 추출하여 정규화 및 추상화를 수행하며, 정규화 및 추상화된 복수의 함수를 취약 코드 클론 집합과 비교하여, 소프트웨어에 대한 코드 클론 여부를 판단하되, 취약 코드 클론 집합은 복수의 타 소프트웨어에 포함된 함수로부터 추출된 후 정규화 및 추상화되어 저장된 복수의 취약 코드를 포함한다.

[0009] 또한, 본 발명의 제 2 측면에 따른 코드 클론 탐지 장치에서의 소프트웨어의 코드 클론 탐지 방법은 소프트웨어에 대응하는 소스 코드로부터 복수의 함수를 추출하는 단계; 추출된 복수의 함수에 대한 정규화 및 추상화를 수행하는 단계; 및 정규화 및 추상화된 복수의 함수를 취약 코드 클론 집합과 비교하여, 소프트웨어에 대한 코드 클론 여부를 판단하는 단계를 포함한다. 이때, 취약 코드 클론 집합은 복수의 타 소프트웨어에 포함된 함수로부터 추출된 후 정규화 및 추상화되어 저장된 복수의 취약 코드를 포함한다.

발명의 효과

[0010] 본 발명은 소프트웨어에 포함된 소스 코드에 대한 정규화 및 추상화를 수행하므로, 소스 코드에 포함된 함수 및

변수의 이름과 자료형 등을 수정한 경우에도 코드 클론을 탐지할 수 있다.

[0011] 또한, 본 발명은 딕셔너리 자료구조에 기초하여, 취약 코드 클론 집합 및 해당 소프트웨어에 대한 정수형의 키를 먼저 비교하고, 키 검색을 통하여 축소된 검색 범위에 포함된 해시 값의 비교를 수행하여, 최종적으로 코드 클론을 탐지하므로 시간 복잡도가 작아질 수 있다. 즉, 본 발명은 상수 시간 내에 코드 클론을 탐지할 수 있으므로, 대용량의 오픈 소스 소프트웨어의 코드 클론을 빠르고 효율적으로 탐지할 수 있다.

도면의 간단한 설명

[0012] 도 1은 본 발명의 일 실시예에 따른 소프트웨어의 코드 클론 탐지 장치의 블록도이다.

도 2는 본 발명의 일 실시예에 따른 소프트웨어의 소스 코드에 대한 예시도이다.

도 3은 본 발명의 일 실시예에 따른 코드 클론 탐지 과정의 예시도이다.

도 4는 본 발명의 일 실시예에 따른 함수의 추상화의 예시도이다.

도 5는 본 발명의 일 실시예에 따른 코드 클론 탐지 장치의 소프트웨어의 코드 클론 탐지 방법에 대한 순서도이다.

발명을 실시하기 위한 구체적인 내용

[0013] 아래에서는 첨부한 도면을 참조하여 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자가 용이하게 실시할 수 있도록 본 발명의 실시예를 상세히 설명한다. 그러나 본 발명은 여러 가지 상이한 형태로 구현될 수 있으며 여기에서 설명하는 실시예에 한정되지 않는다. 그리고 도면에서 본 발명을 명확하게 설명하기 위해서 설명과 관계없는 부분은 생략하였으며, 명세서 전체를 통하여 유사한 부분에 대해서는 유사한 도면 부호를 붙였다.

[0014] 명세서 전체에서, 어떤 부분이 다른 부분과 "연결"되어 있다고 할 때, 이는 "직접적으로 연결"되어 있는 경우뿐 아니라, 그 중간에 다른 소자를 사이에 두고 "전기적으로 연결"되어 있는 경우도 포함한다. 또한, 어떤 부분이 어떤 구성요소를 "포함"한다고 할 때, 이는 특별히 반대되는 기재가 없는 한 다른 구성요소를 제외하는 것이 아니라 다른 구성요소를 더 포함할 수 있는 것을 의미한다.

[0015] 코드 클로닝은 다른 소프트웨어의 소스 코드의 일부 또는 전부를 복제하는 것을 의미한다. 코드 클로닝은 크게 4가지 유형으로 구분할 수 있다.

[0016] 코드 클로닝의 첫 번째 유형은 소스 코드를 수정 없이 그대로 복제하는 것이다. 이때, 원본 소스 코드 및 코드 클론(code clone) 된 소스 코드는 정확히 일치하게 된다.

[0017] 코드 클로닝의 두 번째 유형은 소스 코드 내의 일부 변수(variable) 및 함수(function) 등의 이름 또는 자료형(type)을 수정하는 것이다. 그러므로 원본 소스 코드 및 코드 클론된 소스 코드는 자료형, 식별자(identifier), 주석(comment) 및 화이트스페이스(white space)를 제외한 다른 구문이 일치하게 된다.

[0018] 코드 클로닝의 세 번째 유형은 소스 코드의 일부 변수 또는 함수의 이름을 수정하고, 구조를 보정하는 것이다. 이때, 구조의 보정은 소스 코드의 일부를 삽입 또는 삭제하거나 일부 문장을 재배열하는 것일 수 있다. 그러므로 원본 소스 코드 및 코드 클론된 소스 코드는 일부가 만이 일치할 수 있다.

[0019] 또한, 코드 클로닝의 네 번째 유형은 의미적 복제(sematic clone)이다. 이때, 원본 소스 코드 및 코드 클론된 소스 코드는 서로 구문이 상이하다. 그러나 원본 소스 코드 및 코드 클론된 소스 코드는 기능적으로 동일하므로 같은 일을 수행할 수 있다.

[0020] 소스 코드의 일부분의 구조를 보정하거나, 소스 코드를 의미적으로 복제하는 세 번째 유형 및 네 번째 유형을 수행하기 위해서는 원본 소스 코드에 대한 충분한 분석 및 추가한 구문에 대한 검증 및 디버깅 시간이 소요되게 된다. 그러므로 세 번째 유형 및 네 번째 유형을 수행하기 위해서는 많은 시간이 소요되며, 높은 수준의 프로그래밍 기술이 필요하다.

[0021] 즉, 일반적으로 코드 클로닝은 소스 코드의 개발에 소요되는 시간 및 노력을 절약하기 위하여 수행되므로, 첫 번째 유형 또는 두 번째 유형에 해당한다. 그러므로 본 발명의 일 실시예에 따른 소프트웨어의 코드 클론 탐지 장치(100)는 첫 번째 유형 및 두 번째 유형의 코드 클론을 대상으로 한다. 즉, 코드 클론 탐지 장치(100)는 코드 클로닝을 수행하는 사용자가 소스 코드에 포함된 특정 함수를 그대로 복제하거나, 특정 함수에 포함된 함수 이름, 변수의 자료형 및 변수의 이름을 변경하여 복제하는 코드 클론을 탐지할 수 있다.

[0022] 이하에서는 C 언어 및 C++ 언어 계열의 프로그래밍 언어를 이용하여 개발된 소프트웨어를 예를 들어 본 발명의 일 실시예에 따른 코드 클론 탐지 장치(100) 및 코드 클론 탐지 장치(100)의 코드 클론 탐지 방법을 설명한다. 그러나 프로그래밍 언어는 이에 한정된 것이 아니며, C 언어 및 C++ 언어 계열의 프로그래밍 언어 외의 JAVA, C#, Python 및 BASIC 등과 같은 다양한 프로그래밍 언어를 포함할 수 있다.

[0023] 다음은 도 1 내지 도 4를 참조하여, 본 발명의 일 실시예에 따른 소프트웨어의 코드 클론 탐지 장치(100)를 설명한다.

[0024] 도 1은 본 발명의 일 실시예에 따른 소프트웨어의 코드 클론 탐지 장치(100)의 블록도이다.

[0025] 코드 클론 탐지 장치(100)는 소프트웨어에 포함된 소스 코드로부터 오픈 소스 소프트웨어에 대한 코드 클론을 감지한다.

[0026] 일반적으로 소프트웨어는 하나 이상의 파일(file)을 포함할 수 있다. 파일은 함수(function), 변수(variable) 및 상수(constant)등을 포함할 수 있다.

[0027] 함수는 실제로 일(task)을 수행하는 명령어, 상수 및 변수를 포함할 수 있다. 이때, 함수는 소프트웨어가 개발되는 프로그래밍 환경에 따라, 모듈(module), 메소드(method) 및 프로시저(procedure) 등이 될 수 있으나, 이에 한정된 것은 아니다.

[0028] 예를 들어, C 언어 또는 C++언어에서 함수는 헤더(header) 및 메인(main)으로 구분된다. 이때, 헤더는 반환되는 값의 자료형, 함수 이름 및 해당 함수로 입력되는 복수의 인자(parameter)의 자료형을 포함한다. 또한, 메인은 함수가 하는 구체적인 일이 명시된 하나 이상의 라인(line)을 포함한다. 복수의 라인을 포함하는 경우, 메인은 중괄호('{', '}')을 포함하며, 중괄호('{', '}') 내에 복수의 라인을 포함할 수 있다.

[0029] 또한, C 언어 또는 C++ 언어에서 하나의 라인은 세미 클론(';')으로 구분될 수 있다. 즉, 개행 문자('')의 출현 여부와 상관없이, 세미클론(';')이 나타날 때까지 포함된 문자는 하나의 라인이 될 수 있다.

[0030] 코드 클론 탐지 장치(100)는 소프트웨어의 소스 코드에 포함된 복수의 함수에 기초하여, 코드 클론을 탐지할 수 있다. 이때, 코드 클론 탐지 장치(100)는 메모리(110), 데이터베이스(120) 및 프로세서(130)를 포함할 수 있다. 도 1의 코드 클론 탐지 장치(100)는 본 발명의 하나의 구현 예에 불가하다. 그러므로 코드 클론 탐지 장치(100)는 도 1에 도시된 구성요소를 기초로 여러 가지로 변형이 가능하다.

[0031] 메모리(110)는 소프트웨어에서 코드 클론을 탐지하는 프로그램이 저장된다. 이때, 메모리(110)는 전원이 공급되지 않아도 저장된 정보를 계속 유지하는 비휘발성 저장장치 및 저장된 정보를 유지하기 위하여 전력이 필요한 휘발성 저장장치를 통칭하는 것이다.

[0032] 데이터베이스(120)는 코드 클론을 감지하기 위한 취약 코드 클론 집합을 저장할 수 있다. 이때, 데이터베이스(120)는 코드 클론 탐지 장치(100)와 연결되었거나, 코드 클론 탐지 장치(100)에 탑재된 것일 수 있으나, 이에 한정된 것은 아니다.

[0033] 프로세서(130)는 소프트웨어의 소스 코드에 포함된 복수의 함수와 데이터베이스(120)에 저장된 취약 코드 클론 집합에 저장된 함수와의 비교를 통하여, 소프트웨어의 코드 클론 여부를 판단할 수 있다. 이때, 취약 코드 클론 집합은 기 수집된 복수의 타 소프트웨어에 포함된 함수로부터 추출된 소스 코드를 포함할 수 있다. 또한, 취약 코드 클론 집합은 취약 코드를 포함하는 소프트웨어로부터 추출된 소스 코드를 포함하거나, 라이선스 규약 위배 여부를 판단할 미리 정의된 오픈 소스 소프트웨어로부터 추출된 소스 코드를 포함할 수 있으나, 이에 한정된 것은 아니다.

[0034] 도 2는 본 발명의 일 실시예에 따른 소프트웨어의 소스 코드에 대한 예시도이다.

[0035] 도 2를 참조하면, 소프트웨어의 소스 코드는 제 1 함수(200) 및 제 2 함수(210)를 포함할 수 있다. 이때, 소스 코드에 포함된 복수의 함수는 각각의 반환되는 값의 자료형, 함수 이름 및 해당 함수로 입력되는 복수의 인자의 자료형을 포함하는 헤더(201, 211) 및 해당 함수가 실제 수행하는 일을 정의한 바디(202, 212)를 포함할 수 있다.

[0036] 예를 들어, 제 1 함수(200)는 헤더(201)로 "func1 (bar)" 를 포함한다. 그리고 제 1 함수(200)는 바디(202)로 헤더 아래의 중괄호('{', '}')을 내의 4개의 라인 "foo = bar;", "foo ++;", "bar = fun2(foo);" 및 "return bar;"를 포함할 수 있다.

[0037] 또한, 제 2 함수(210)는 헤더(211)로 "func2 (param)"를 포함한다. 그리고 제 2 함수(210)는 바디(212)로 헤더 아래의 중괄호('{', '}')을 내의 2개의 라인 "if(param) {while (param) { param -- ;}}" 및 "return 0;"를 포함할 수 있다.

[0038] 도 3은 본 발명의 일 실시예에 따른 코드 클론 탐지 과정의 예시도이다.

[0039] 프로세서(130)는 소스 코드로부터 복수의 함수를 검색하여 추출한다(S300). 그리고 프로세서(130)는 추출된 복수의 함수에 대한 정규화 및 추상화를 수행할 수 있다(S300, S310). 이때, 프로세서(130)는 정규식(regular expression; regex)을 이용할 수 있다.

[0040] 정규식은 특정한 규칙에 기초하여, 문자열의 집합을 표현하는데 사용하는 형식 언어이다. 정규식은 문자열 내의 특정 문자를 추출하거나, 제거하기 위하여 사용될 수 있다. 또한, 정규식은 문자열 내에서 변수 이름 및 변수의 자료형을 추출하거나, 추상화하기 위하여 사용할 수 있다. 예를 들어, 프로세서(130)는 "(₩**₩w+[:]*₩w+)"을 이용하여, 함수 내의 함수 명을 추출할 수 있다.

[0041] 구체적으로 함수는 프로그래밍 언어에 따라, 미리 정해진 포맷(format)이 존재한다. 예를 들어, 앞에서 설명한 바와 같이, C 언어 및 C++언어에서 함수는 헤더 및 메인이 존재한다. C 언어 및 C++ 언어에서 함수의 헤더는 함수가 반환하는 값의 자료형, 함수 이름과 소괄호('(', ')')로 둘러싸인 0개 이상의 인자를 포함할 수 있다. 이때, 함수가 반환하는 값이 없는 경우, 함수가 반환하는 값의 자료형은 생략되거나, void로 표현될 수 있다.

[0042] 또한, C 언어 및 C++ 언어에서 함수의 바디는 헤더 다음에 위치할 수 있다. 예를 들어, 헤더 다음에 중괄호('{')가 있는 경우, 함수의 바디는 해당 중괄호가 끝날 때('}')까지가 포함된 하나 이상의 라인이 될 수 있다. 또한, 헤더 다음에 중괄호가 없는 경우, 함수의 바디는 헤더 다음의 한 개의 라인이 될 수 있다.

[0043] 그러므로 프로세서(130)는 코드 클론을 탐지하는 프로그래밍 언어에 대응하여, 미리 정의된 함수 추출을 위한 정규식에 기초하여, 함수를 추출할 수 있다.

[0044] 도 2를 참조하면, 프로세서(130)는 미리 정해진 정규식에 기초하여, 제 1 함수(200)인 "func1"함수에서 함수의 헤더(201)로 "func1 (bar)"를 추출하며, 함수의 바디(202)로 함수 헤더(201) 바로 아래의 '{' 및 '}'로 둘러싸인 부분을 추출할 수 있다. 또한, 프로세서(130)는 제 2 함수(210)인 "func2"함수에서 함수의 헤더(211)로 "func2 (param)"을 추출하며, 함수의 바디(212)로 함수 헤더(211) 바로 아래의 '{' 및 '}'로 둘러 쌓인 부분을 추출할 수 있다.

[0045] 그리고 프로세서(130)는 추출된 복수의 함수에 대한 정규화 및 추상화를 수행할 수 있다. 도 3을 참조하면, 프로세서(130)는 복수의 함수에 대한 정규화를 수행한 이후, 추상화를 수행하는 것으로 도시되어 있다. 그러나 도 3은 본 발명의 하나의 구현 예에 불가하다. 그러므로 정규화 및 추상화의 순서는 구현 예에 따라, 다양하게 변경이 가능하다. 즉, 프로세서(130)는 도 3과 같이, 복수의 함수에 대한 정규화를 수행한 이후, 추상화를 수행할 수 있다. 또한, 프로세서(130)는 복수의 함수에 대한 추상화를 수행한 이후, 정규화를 수행할 수 있다.

[0046] 이때, 정규화는 함수로부터 헤더 및 바디를 추출하고, 추출된 바디에 포함된 코드 클론에 필요 없는 부분을 제거하는 것일 수 있다. 구체적으로 앞에서 설명한 바와 같이, 프로세서(130)는 미리 정해진 포맷에 기초하여 함수로부터 헤더 및 메인이 분리되면, 메인에 포함된 복수의 라인에 대하여, 코드 클론 분석에 필요 없는 부분을 제거할 수 있다. 이때, 코드 클론 분석에 필요 없는 부분은 화이트스페이스(whitespace) 문자 및 주석(comment) 등이 될 수 있다.

[0047] 프로세서(130)는 앞에서 설명한 정규식을 이용하여, 함수로부터 헤더 및 메인을 분리하고, 화이트스페이스 문자 또는 주석을 제거할 수 있다. 이때, 화이트스페이스 문자는 공백(space), 탭(tab) 문자 및 개행(new line) 문자를 포함할 수 있다.

[0048] 프로세서(130)는 정규화된 함수에 대한 추상화를 수행할 수 있다.

[0049] 이때, 추상화는 함수 내의 변수 이름 및 변수 자료형을 일반화(generalization)하여, 변수 이름 및 변수 자료형을 변경한 경우에도 코드 클론을 탐지하기 위하여 수행될 수 있다. 실제로, 소스 코드를 클로닝하는 사용자가 변수의 이름 또는 변수의 자료형을 수정하거나, 변수의 자료형에 한정자(qualifier)를 추가하는 방식으로 변수의 자료형을 보정하는 것은 소스 코드의 다른 부분을 수정하는 것에 비해 매우 쉽다. 그러므로 이러한 경우는 매우 자주 발생할 수 있다.

[0050] 이와 같이, 프로세서(130)는 두 개의 함수에 포함된 변수의 이름 또는 변수의 자료형이 상이하더라도 실제 일을

수행하는 소스 코드가 동일한 경우, 두 함수를 코드 클론으로 판단할 수 있다. 이를 위하여, 프로세서(130)는 변수 이름 및 변수의 자료형을 추상화하여, 변수 이름 및 변수의 자료형을 수정하는 코드 클론을 용이하게 탐지할 수 있다. 이때, 프로세서(130)는 앞에서 설명한 정규식을 활용하여, 함수의 추상화를 수행할 수 있다. 함수의 추상화는 도 4를 참조하여, 상세하게 설명한다.

[0051] 도 4는 본 발명의 일 실시예에 따른 함수의 추상화의 예시도이다.

[0052] 도 4의 (a)는 함수의 추상화를 수행하기 이전의 원본 함수이다. 해당 함수의 헤더는 실수형 자료형(float) 배열(array)인 "arr"과 정수형 자료형(int)인 "len"을 인자로 포함한다. 그러므로 프로세서(130)는 함수의 헤더에 포함된 인자의 변수 이름을 미리 정해진 식별자로 수정하는 추상화를 수행할 수 있다. 예를 들어, 프로세서(130)는 도 4의 (b)와 같이, 함수의 인자에 포함된 변수의 이름을 함수의 인자에 포함된 변수에 대응하여 미리 정해진 식별자인 "FPARAM"으로 추상화할 수 있다.

[0053] 함수의 인자에 대한 추상화를 수행한 이후, 프로세서(130)는 함수의 바디에 포함된 변수의 자료형에 대하여 추상화를 수행할 수 있다. 먼저, 프로세서(130)는 변수의 자료형에 한정자를 삭제할 수 있다. 이때, 한정자는 short, long, signed, unsigned, static, extern, volatile, auto 및 resister 중 하나 이상이 될 수 있으나, 이에 한정된 것은 아니다. 그리고 프로세서(130)는 int 및 char, 등과 같은 정수형 자료형, float 및 double 등과 같은 실수형 자료형, bool 등 과 같은 논리 자료형, 및 void 자료형 등의 자료형을 자료형에 대응하는 미리 정해진 식별자로 변경할 수 있다.

[0054] 예를 들어, 프로세서(130)는 도 4의 (c)와 같이, 함수의 바디에 포함된 변수의 자료형의 한정자인 "static"을 제거하고, "float" 자료형인 "sum" 변수 및 "int" 자료형인 "i" 변수의 자료형을 자료형에 대응하는 미리 정해진 식별자인 "DTYPE"으로 추상화 할 수 있다.

[0055] 마지막으로 프로세서(130)는 함수의 바디에 포함된 변수의 이름을 바디에 포함된 변수에 이름에 대응하는 미리 정해진 식별자에 기초하여 추상화할 수 있다. 예를 들어, 도 4의 (d)와 같이, 프로세서(130)는 함수의 바디에 포함된 변수의 이름을 바디에 포함된 변수에 이름에 대응하는 미리 정해진 식별자인 "LVAR"로 추상화할 수 있다.

[0056] 원본 함수인 도 4의 (a)와 추상화를 완료한 도 4의 (d)를 참조하면, 두 함수는 함수의 인자 이름 및 함수 내의 변수의 자료형과 이름이 상이하다. 그러나 두 함수는 실수형 배열 및 정수형 값을 인자로 입력 받고, 실수형 배열에 대한 평균을 산출하여 화면에 출력하는 동일한 일을 수행하게 된다.

[0057] 소프트웨어에 포함된 복수의 함수에 대한 정규화 및 추상화를 수행한 이후, 프로세서(130)는 정규화 및 추상화된 함수와 취약 코드 클론 집합을 비교하기 위하여, 딕셔너리(dictionary) 자료구조를 이용하여 표현할 수 있다(S320).

[0058] 딕셔너리 자료구조는 키와 키와 대응되는 하나 이상의 값을 포함할 수 있다. 이때, 하나 이상의 값은 배열 또는 집합(set) 형태로 포함될 수 있다. 또한, 딕셔너리 자료구조는 해시마크 딕셔너리(hashmark dictionary)일 수 있다.

[0059] 구체적으로 프로세서(130)는 소프트웨어에 포함된 복수의 함수 각각에 대하여, 추상화된 함수 바디의 길이를 키(key)로 추상화된 함수에 대응하는 해시 값(hash value)을 값(value)으로 설정하고, 딕셔너리 자료구조에 추가할 수 있다. 이때, 프로세서(130)는 함수의 바디로부터 추출된 문자열 및 미리 정의된 해시 함수(hash function)에 기초하여, 키에 대응하는 해시 값을 추출할 수 있다.

[0060] 예를 들어, 해시 함수는 다음 세 가지 조건을 만족하도록 정해질 수 있다.

[0061] 조건 1: 해시 함수를 통하여 산출되는 해시 값의 충돌(collision)을 최소화함.

[0062] 조건 2: 최소의 딕셔너리를 생성하기 위하여, 해시 값이 적은 수의 비트를 포함하도록 해시 함수를 설정함.

[0063] 조건 3: 해시 값을 생성하는데 소요되는 시간의 복잡도가 O(n)이 되도록 해시 함수를 설정함.

[0064] 프로세서(130)는 조건 1에 따라, [수학식 1]과 같이, 충돌 확률 $P_{\text{collision}}$를 산출할 수 있다. 이때, 충돌 확률 $P_{\text{cillision}}$는 충돌하지 않은 확률 $P_{\text{uncollision}}$과 상호 배타적(mutually exclusive)이다.

수학식 1

[0065] $$P_{collision} = 1 - P_{nocollision}$$

[0066] 이때, 샘플 스페이스(sample space) S에서 i번째 샘플(sample)이 앞서 산출된 해시 값과 충돌이 일어나지 않을 이벤트를 A_i라고 할 때, 1부터 i번째 샘플까지 충돌이 발생하지 않을 확률은 [수학식 2]와 같다.

수학식 2

[0067] $$P_{nocollision\ in\ i\ samples} = P(A_1)P(A_2|A_1)\cdots P(A_i|A_{i-1}) = 1 \times \left(1 - \frac{1}{S}\right) \times \cdots \times \left(1 - \frac{i-1}{S}\right)$$

[0068] 프로세서(130)는 [수학식 2]에 기초하여, N개의 해시 함수를 통하여, 산출 가능한 값 및 k 개의 메시지에 대한 충돌이 발생하지 않을 확률을 [수학식 3]과 같이 산출할 수 있다. 이때, N 및 k 는 자연수이며, $N >> k >> 1$이 될 수 있다.

수학식 3

[0069] $$P_{nocollision} = 1 \times \left(1 - \frac{1}{N}\right) \times \left(1 - \frac{2}{N}\right) \times \cdots \times \left(1 - \frac{k-1}{N}\right)$$

[0070] 또한, 테일러 급수(Taylor series)에 기초하여, x가 0에 수렴할 때, $1 - x \simeq e^{-x}$ 이므로, [수학식 3]은 [수학식 4]와 같이 표현할 수 있다.

수학식 4

[0071] $$P_{nocollision} = 1 \times e^{\frac{-1}{N}} \times e^{\frac{-2}{N}} \times \cdots \times e^{\frac{-(k-1)}{N}} = e^{\frac{-k(k-1)}{2N}}$$

[0072] 프로세서(130)는 [수학식 1] 및 [수학식 4]에 기초하여, 충돌이 발생할 확률 $P_{collision}$를 [수학식 5]와 같이, 산출할 수 있다. 또한, 프로세서(130)는 [수학식 5] 및 테일러 급수에 기초하여, [수학식 6]을 산출할 수 있다.

수학식 5

[0073] $$P_{collision} = 1 - P_{nocollision} = 1 - e^{\frac{-k(k-1)}{2N}}$$

수학식 6

[0074]
$$P_{collision} = 1 - e^{\frac{-k(k-1)}{2N}} \simeq \frac{k(k-1)}{2N} \simeq \frac{k^2}{2N}$$

[0076] 예를 들어, [수학식 6]를 참조하면, 128bit의 해시 함수를 사용할 경우 해시 값은 2^{37}번 중 한번 충돌될 수 있다. 즉, 충돌 확률은 $2.776e^{-15}$%가 될 수 있다. 즉, 해시 함수는 조건 1에 따라, 충돌 확률을 최소화하도록 128bit의 해시 함수를 사용할 수 있다.

[0077] 또한, 프로세스는 알려진 해시 함수에 기초하여, 키에 대응하는 해시 값을 산출할 수 있다. 이때, 해시 함수는 MD4(message digest 4), MD5(message digest 5), CityHash, MurmurHash 및 SpookyHash 등과 같이 알려진 해시 함수이거나, 소프트웨어 코드 클론을 탐지하기 위하여 개발된 사용자 정의 해시 함수일 수 있으나 이에 한정된 것은 아니다.

[0078] 예를 들어, 프로세서(130)는 조건 1 내지 3에 기초하여, 알려진 해시 함수 중 시간 복잡도가 작으며, 128 bit 해시 함수인 MD5를 이용할 수 있다. 다시 도 2를 참조하면, 프로세서(130)는 제 1 함수(200)에 대하여, 추상화된 바디의 길이인 39를 키로 설정할 수 있다. 그리고 프로세서(130)는 제 1 함수(200)에 대하여 추상화된 바디 및 MD5 해시 함수에 기초하여, 해시 값을 "8b03 c2a8 ecea 8cc4 9c6d d780 9771 cfd9"로 산출할 수 있다.

[0079] 또한, 프로세서(130)는 제 2 함수(210)에 대하여 추상화된 바디 길이인 39를 키로 설정할 수 있다. 그리고 프로세서(130)는 추상화된 제 2 함수(210)의 바디 및 해시 함수에 기초하여, 해시 값을 "019c e125 de43 45d0 87d7 706f 1482 2bf1"로 설정할 수 있다.

[0080] 프로세서(130)는 해당 소프트웨어에 대응하는 딕셔너리에 제 1 함수(200)에 대한 키 및 해시 값(204)와 제 2 함수(210)에 대한 키 및 해시 값(214)을 저장할 수 있다. 이때, 제 1 함수(200) 및 제 2 함수(210)가 동일한 키가 설정되므로, 실제 딕셔너리에 저장되는 값은 키 39 및 키에 대응하는 해시 값의 집합 {"8b03 c2a8 ecea 8cc4 9c6d d780 9771 cfd9", "019c e125 de43 45d0 87d7 706f 1482 2bf1"}이 될 수 있다.

[0081] 한편, 소프트웨어에 대응하여 추출된 복수의 함수에 대한 딕셔너리가 생성되면, 프로세서(130)는 취약 코드 클론 집합과 비교하여, 해당 소프트웨어에 포함된 코드 클론을 탐지할 수 있다(S330).

[0082] 이때, 취약 코드 클론 집합에 포함된 소스 코드는 정규화 및 추상화되어 저장된 것일 수 있다. 즉, 프로세서(130)는 해당 소프트웨어에 대한 코드 클론을 탐지하기 이전에, 기수집된 소프트웨어로부터 복수의 함수를 추출하고, 추출된 함수에 대한 정규화 및 추상화를 수행할 수 있다. 그리고 프로세서(130)는 정규화 및 추상화된 함수를 딕셔너리 자료구조에 기초하여, 취약 코드 클론 집합에 저장할 수 있다. 이때, 딕셔너리 자료구조는 앞에서 설명한 바와 같이, 해시마크 딕셔너리일 수 있다.

[0083] 또한, 취약 코드 클론 집합은 하나 이상의 소프트웨어에 대하여 각각 생성된 딕셔너리를 포함하는 것이거나, 하나 이상의 소프트웨어에 대하여 생성된 하나의 딕셔너리일 수 있다.

[0084] 그리고 프로세스는 해당 소프트웨어에 포함된 함수를 포함하는 딕셔너리가 생성되면, 생성된 딕셔너리 및 취약 코드 클론 집합의 키를 비교할 수 있다. 그리고 프로세서(130)는 복수의 함수에 대한 딕셔너리의 키 및 취약 코드 클론 집합의 키가 일치하는 경우, 해당 키에 포함된 각각의 값을 비교할 수 있다.

[0085] 다시 도 2를 참조하면, 프로세서(130)는 취약 코드 클론 집합에 대하여, 해당 소프트웨어에 대응하는 딕셔너리에 포함된 키 39와 일치하는 키를 검색할 수 있다. 그리고 프로세서(130)는 취약 코드 클론 집합 중 키가 39인 경우가 존재하지 않으면, 프로세서(130)는 해당 소프트웨어가 코드 클론이 아님을 판단할 수 있다.

[0086] 만약, 취약 코드 클론 집합 중 키가 39인 경우가 존재하면, 프로세서(130)는 키 39에 대응하는 해시 값과 제 1 함수(200) 및 제 2 함수(210)로부터 추출된 해시 값 {"8b03...", "019c..."}을 비교할 수 있다. 그리고 취약 코드 클론 집합 중 키 39에 대응하는 해시 값에 제 1 함수(200) 및 제 2 함수(210)로부터 추출된 해시 값이 존재하지 않은 경우, 프로세서(130)는 해당 소프트웨어가 코드 클론이 아님을 판단할 수 있다. 또한, 취약 코드

클론 집합 중 키 39에 대응하는 해시 값에 제 1 함수(200) 및 제 2 함수(210)로부터 추출된 해시 값이 존재하는 경우, 프로세서(130)는 해당 소프트웨어가 코드 클론임을 판단할 수 있다.

[0087] 이와 같이, 프로세서(130)는 딕셔너리 자료구조에 기초하여, 취약 코드 클론 집합의 키 값과 소프트웨어로부터 추출된 함수에 대응하는 키 값을 먼저, 비교할 수 있다. 그러므로 프로세서(130)는 취약 코드 클론 집합에 포함된 복수의 소스 코드와 소프트웨어에 포함된 복수의 소스 코드를 각각 비교하지 않을 수 있다. 즉, 프로세서(130)는 키 값이 일치하는 소스 코드만을 비교할 수 있으므로 검색 범위(search space)를 줄이는 효과가 있다. 또한, 키 값은 정수이므로, 프로세서(130)는 문자열의 비교보다 빠르게 키 값을 검색할 수 있다.

[0088] 한편, 프로세서(130)는 해당 소프트웨어에 대한 코드 클론 여부를 판단한 이후, 해당 소프트웨어로부터 추출된 딕셔너리를 취약 코드 클론 집합에 추가할 수 있다. 그리고 프로세서(130)는 향후 해당 소프트웨어로부터 추출된 딕셔너리가 추가된 취약 코드 클론 집합에 기초하여, 타 소프트웨어에 대한 코드 클론 여부를 판단할 수 있다.

[0089] 다음은 도 5를 참조하여, 본 발명의 일 실시예에 따른 코드 클론 탐지 장치(100)에서의 소프트웨어의 코드 클론 탐지 방법을 설명한다.

[0090] 도 5는 본 발명의 일 실시예에 따른 코드 클론 탐지 장치(100)의 소프트웨어의 코드 클론 탐지 방법에 대한 순서도이다.

[0091] 코드 클론 탐지 장치(100)는 소프트웨어에 대응하는 소스 코드로부터 복수의 함수를 추출한다(S500).

[0092] 그리고 코드 클론 탐지 장치(100)는 추출된 복수의 함수에 기초하여, 정규화 및 추상화를 수행한다(S510).

[0093] 코드 클론 탐지 장치(100)는 추상화된 복수의 함수를 취약 코드 클론 집합과 비교하여, 소프트웨어에 대한 코드 클론 여부를 판단한다(S520). 이때, 취약 코드 클론 집합은 복수의 타 소프트웨어에 포함된 함수로부터 추출된 후 정규화 및 추상화되어 저장된 복수의 취약 코드를 포함한다.

[0094] 한편, 코드 클론 탐지 장치(100)는 추출된 복수의 함수에 기초하여, 정규화 및 추상화를 수행한 이후, 정규화 및 추상화된 복수의 함수에 기초하여, 소프트웨어에 대한 딕셔너리를 생성할 수 있다.

[0095] 이때, 코드 클론 탐지 장치(100)는 정규화 및 추상화된 함수의 문자열 길이를 딕셔너리의 키로 설정할 수 있다. 또한, 코드 클론 탐지 장치(100)는 상기 정규화 및 추상화된 함수를 딕셔너리의 값으로 설정할 수 있다.

[0096] 그리고 코드 클론 탐지 장치(100)는 해당 소프트웨어에 대한 코드 클론 여부를 판단하기 위하여, 생성된 딕셔너리 및 취약 코드 클론 집합을 비교할 수 있다.

[0097] 이때, 취약 코드 클론 집합 및 해당 소프트웨어에 대응하여 생성된 딕셔너리는 딕셔너리 자료구조에 기초하여 생성된 것일 수 있다.

[0098] 그러므로 코드 클론 탐지 장치(100)는 취약 코드 클론 집합의 키와 해당 소프트웨어에 대응하여 생성된 딕셔너리의 키를 비교할 수 있다. 그리고 코드 클론 탐지 장치(100)는 키값이 동일한 경우, 취약 코드 클론 집합의 키에 대응하는 값과 해당 소프트웨어의 딕셔너리의 키에 대응하는 값을 비교할 수 있다.

[0099] 그리고 최종적으로 각각의 값이 동일한 경우, 프로세서(130)는 해당 소프트웨어의 코드 클론을 탐지할 수 있다.

[0100] 본 발명의 일 실시예에 따른 소프트웨어에 대한 코드 클론 탐지 장치(100) 및 방법은 소프트웨어에 포함된 소스 코드에 대한 정규화 및 추상화를 수행하므로, 소스 코드에 포함된 함수 및 변수의 이름과 자료형 등을 수정한 경우에도 코드 클론을 탐지할 수 있다.

[0101] 또한, 소프트웨어에 대한 코드 클론 탐지 장치(100) 및 방법은 딕셔너리 자료구조에 기초하여, 취약 코드 클론 집합 및 해당 소프트웨어에 대한 정수형의 키를 먼저 비교하고, 키 검색을 통하여 축소된 검색 범위에 포함된 해시 값의 비교를 수행하여, 최종적으로 코드 클론을 탐지하므로 시간 복잡도가 상수 시간 내로 짧아질 수 있다. 즉, 소프트웨어에 대한 코드 클론 탐지 장치(100) 및 방법은 상수 시간 내에 코드 클론을 탐지할 수 있으므로, 대용량의 오픈 소스 소프트웨어의 코드 클론을 빠르고 효율적으로 탐지할 수 있다.

[0102] 본 발명의 일 실시예는 컴퓨터에 의해 실행되는 프로그램 모듈과 같은 컴퓨터에 의해 실행 가능한 명령어를 포함하는 기록 매체의 형태로도 구현될 수 있다. 컴퓨터 판독 가능 매체는 컴퓨터에 의해 액세스될 수 있는 임의의 가용 매체일 수 있고, 휘발성 및 비휘발성 매체, 분리형 및 비분리형 매체를 모두 포함한다. 또한, 컴퓨터 판독가능 매체는 컴퓨터 저장 매체 및 통신 매체를 모두 포함할 수 있다. 컴퓨터 저장 매체는 컴퓨터 판독가능

명령어, 데이터 구조, 프로그램 모듈 또는 기타 데이터와 같은 정보의 저장을 위한 임의의 방법 또는 기술로 구현된 휘발성 및 비휘발성, 분리형 및 비분리형 매체를 모두 포함한다. 통신 매체는 전형적으로 컴퓨터 판독가능 명령어, 데이터 구조, 프로그램 모듈, 또는 반송파와 같은 변조된 데이터 신호의 기타 데이터, 또는 기타 전송 메커니즘을 포함하며, 임의의 정보 전달 매체를 포함한다.

[0103] 본 발명의 방법 및 시스템은 특정 실시예와 관련하여 설명되었지만, 그것들의 구성 요소 또는 동작의 일부 또는 전부는 범용 하드웨어 아키텍처를 갖는 컴퓨터 시스템을 사용하여 구현될 수 있다.

[0104] 전술한 본 발명의 설명은 예시를 위한 것이며, 본 발명이 속하는 기술분야의 통상의 지식을 가진 자는 본 발명의 기술적 사상이나 필수적인 특징을 변경하지 않고서 다른 구체적인 형태로 쉽게 변형이 가능하다는 것을 이해할 수 있을 것이다. 그러므로 이상에서 기술한 실시예들은 모든 면에서 예시적인 것이며 한정적이 아닌 것으로 이해해야만 한다. 예를 들어, 단일형으로 설명되어 있는 각 구성 요소는 분산되어 실시될 수도 있으며, 마찬가지로 분산된 것으로 설명되어 있는 구성 요소들도 결합된 형태로 실시될 수 있다.

[0105] 본 발명의 범위는 상기 상세한 설명보다는 후술하는 특허청구범위에 의하여 나타내어지며, 특허청구범위의 의미 및 범위 그리고 그 균등 개념으로부터 도출되는 모든 변경 또는 변형된 형태가 본 발명의 범위에 포함되는 것으로 해석되어야 한다.

부호의 설명

[0106] 100: 코드 클론 탐지 장치

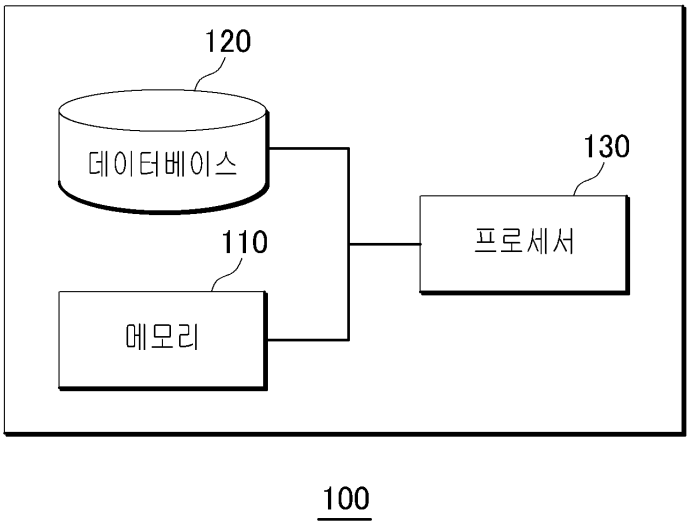
110: 메모리

120: 데이터베이스

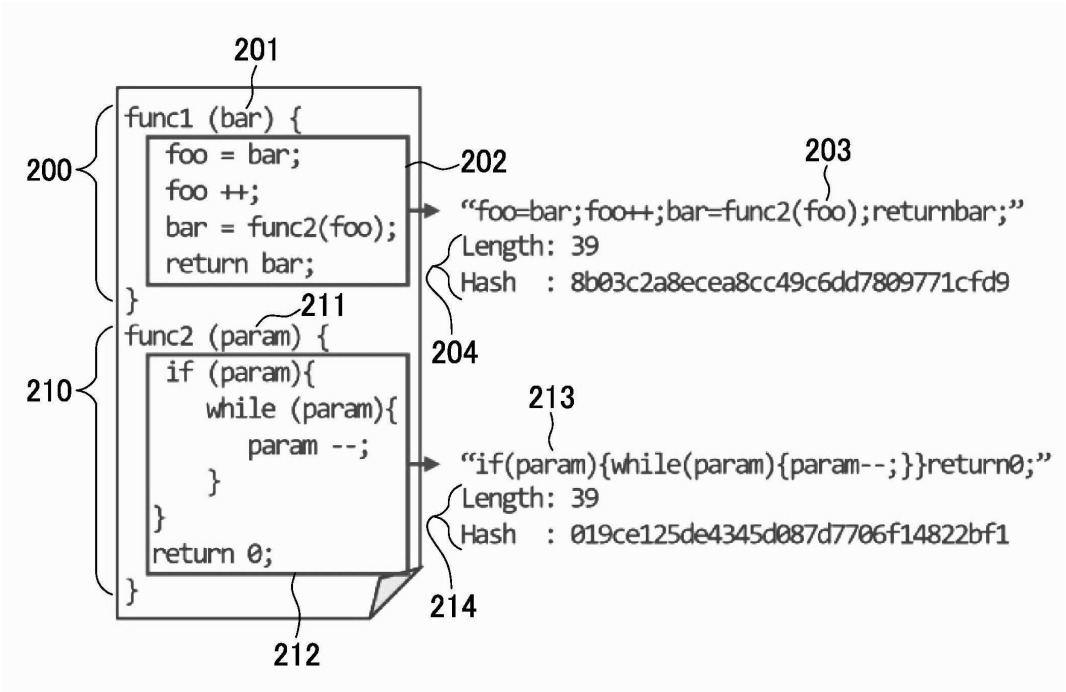
130: 프로세서

도면

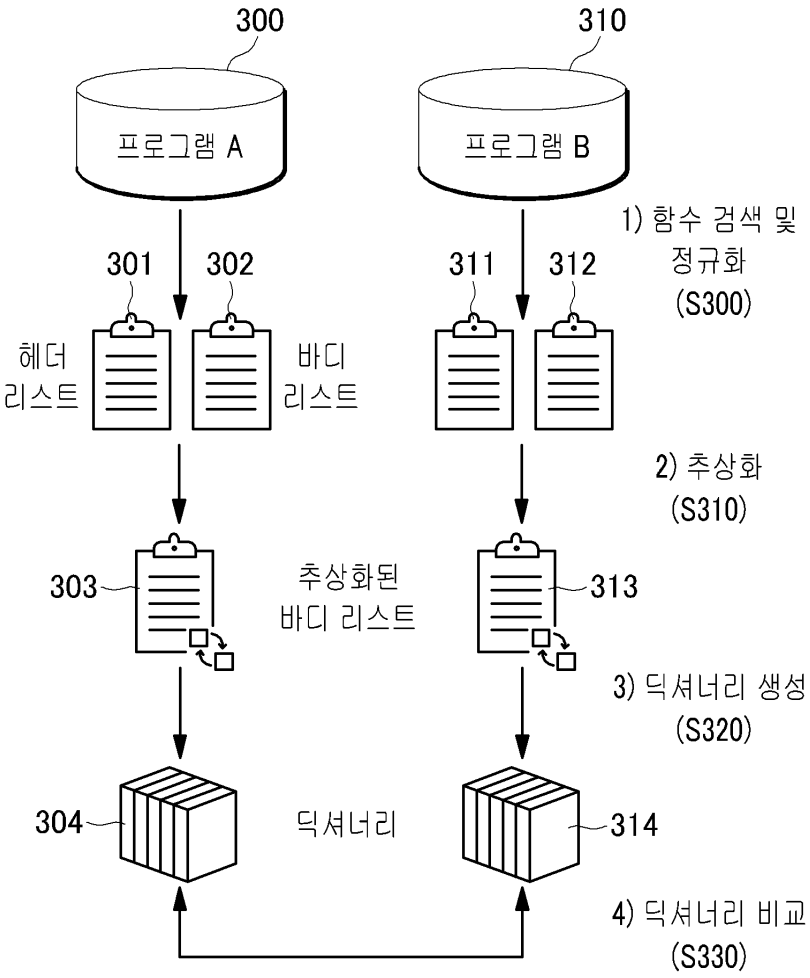
도면1



도면2


201
200
func1 (bar) {
foo = bar;
foo ++;
bar = func2(foo);
return bar;
}
202
203
"foo=bar;foo++;bar=func2(foo);returnbar;"
Length: 39
Hash : 8b03c2a8ecea8cc49c6dd7809771cfd9
204
211
210
func2 (param) {
if (param){
while (param){
param --;
}
}
return 0;
}
213
"if(param){while(param){param--;}}return0;"
Length: 39
Hash : 019ce125de4345d087d7706f14822bf1
214
212

도면3


300
310
프로그램 A
프로그램 B
1) 함수 검색 및
정규화
(S300)
301
302
311
312
헤더
리스트
바디
리스트
2) 추상화
(S310)
303
추상화된
바디 리스트
313
3) 딕셔너리 생성
(S320)
304
딕셔너리
314
4) 딕셔너리 비교
(S330)

도면4

(a)

```
1  void avg (float arr[], int len) {
2     static float sum = 0;
3     int i;
4     for(i = 0; i < len; i++)
5        sum += arr[i];
6     printf("%f", sum / len);
7  } // Level 0: No abstraction
```

(b)

```
1  void avg (float FPARAM[], int FPARAM) {
2     static float sum = 0;
3     int i;
4     for(i = 0; i < FPARAM; i++)
5        sum += FPARAM[i];
6     printf("%f", sum / FPARAM);
7  } // Level 1: Formal PARAMeter abstraction
```

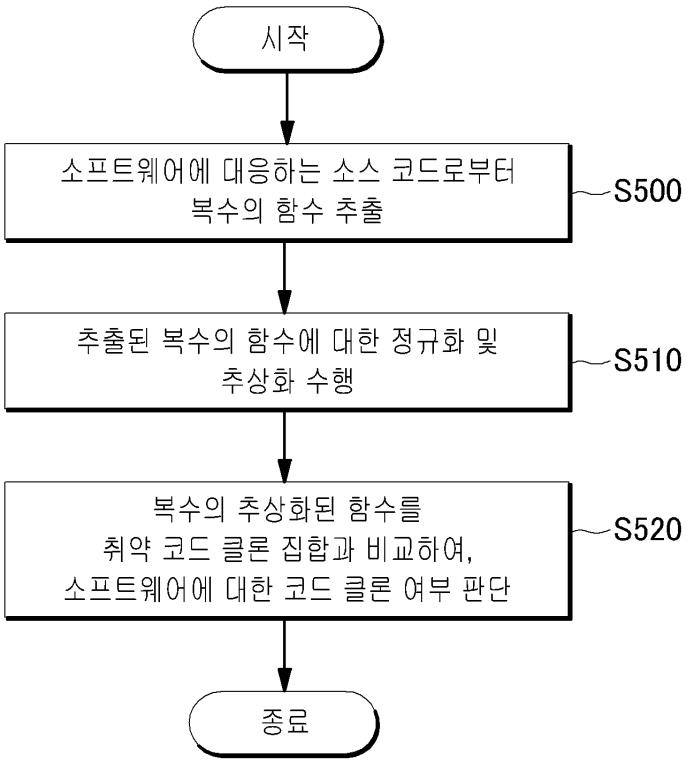
(c)

```
1  void avg (float FPARAM[], int FPARAM) {
2     DTYPE sum = 0;
3     DTYPE i;
4     for(i = 0; i < FPARAM; i++)
5        sum += FPARAM[i];
6     printf("%f", sum / FPARAM);
7  } // Level 2: Data TYPE abstraction
```

(d)

```
1  void avg (float FPARAM[], int FPARAM) {
2     DTYPE LVAR = 0;
3     DTYPE LVAR;
4     for(LVAR = 0; LVAR < FPARAM; LVAR++)
5        LVAR += FPARAM[LVAR];
6     printf("%f", LVAR / FPARAM);
7  } // Level 3: Local VARiable abstraction
```

도면5


시작
소프트웨어에 대응하는 소스 코드로부터 복수의 함수 추출
S500
추출된 복수의 함수에 대한 정규화 및 추상화 수행
S510
복수의 추상화된 함수를 취약 코드 클론 집합과 비교하여, 소프트웨어에 대한 코드 클론 여부 판단
S520
종료