제어 흐름을 고려한 API k-gram 소프트웨어 버스마크*

박희완, 최석우, 임현일, 한태숙
한국과학기술원 전자전산학과 전산학전공
e-mail:{hwpark, swchoi, hilim}@pllab.kaist.ac.kr, han@cs.kaist.ac.kr

A Flow-sensitive API k-gram Based Software Birthmark

Heewan Park, Seokwoo Choi, Hyun-il Lim and Taisook Han
Division of Computer Science, Dept. of EECS, KAIST

요 약

소프트웨어 버스마크는 클래스나 바이너리를 대상으로 고유한 특징을 추출하여 유사도를 비교하는 방법이다. 본 논문에서는 자바의 제어 흐름을 고려한 API k-gram 버스마크 기법을 제안한다. 이 기법은 다른 것으로 대체하기 어려운 자바 표준 API 호출에 대한 시퀀스를 k-gram으로 사용했기 때문에 신뢰도가 높고, 제어 흐름을 반영하여 시퀀스를 추출하기 때문에 난독화에 강하다는 장점이 있다. 본 논문에서 제안하는 버스마크를 기법을 평가하기 위해서 신뢰도와 강인도에 대한 실험을 한 결과 기존의 버스마크보다 신뢰도와 강인도가 우수한 버스마크임을 확인하였다.

1. 서론

최근 인터넷의 발달로 소프트웨어가 쉽게 복제 및 배포될 수 있어서 소프트웨어 도용 사례가 급격히 증가하고 있다. 특히 자바 프로그램의 경우는 이식성이 높아서 도용의 주된 표적이 될 수 있다. 만일 도용이 의심되는 프로그램의 소스 코드를 얻을 수 있다면 소스 표절 검사 기법이 사용될 수 있다. 그러나 일반적으로 프로그램은 컴파일되어 배포되기 때문에 클래스 파일이나 바이너리 파일에 직접 적용할 수 있는 도용 탐지 기법이 필요하다.

소프트웨어 버스마크는 클래스나 바이너리를 대상으로 고유한 특징을 추출하여 유사도를 비교하는 방법이다. 본 논문에서는 자바의 제어 흐름을 고려한 API k-gram 버스마크 기법을 제안하고 효용성을 평가한다.

2. 관련 연구

Tamada는 자바 클래스 파일에 대한 정적 소프트웨어 버스마크[1] 기법을 제안했다. 이 기법은 자바 클래스의 필드 변수에 사용된 상수 값, 자바 메소드 호출 시퀀스, 클래스 상속 구조, 사용된 클래스의 4가지 버스마크로 구성된다. 자바 클래스의 구조적인 특징 위주로 버스마크를 구성하였기 때문에 서로 다른 두 프로그램은 구별하는 능력이 떨어진다는 단점이 있다.

Myles는 k-gram 버스마크[2]를 제안하였다. 여기서 k-gram이란 자바 클래스에서 연속된 k개의 JVM 명령어 시퀀스 집합을 의미한다. 즉, 자바 클래스를 이루고 있는 JVM 명령어를 길이가 k인 조각으로 잘라서 버스마크로 사용한다. k-gram 버스마크는 서로 다른 프로그램을 구별하는 능력은 뛰어나지만 JVM 명령어가 변경되거나 시퀀스가 바뀌는 경우가 발생할 수 있기 때문에 버스마크가 쉽게 손상된다는 단점이 있다.

자바 메소드에 대한 정적 API 트레이스를 사용한 버스마크[3]도 연구가 되었다. API 트레이스 버스마크는 API 메소드로부터 실행 가능한 모든 API 트레이스를 생성하기 때문에 기존의 버스마크와 비교하여 서로 다른 프로그램을 구별하는 능력이 높고 프로그램 변환에도 강인하다는 특징이 있다. 그러나 조건문이 많아질수록 트레이스의 수가 기하급수적으로 증가하는 문제가 발생하기 때문에 프로그램의 크기가 크고 제어 흐름 그래프가 복잡할 경우에는 실행 경로의 수가 급증하여 트레이스 추출 자체가 불가능하다는 단점이 있다.

3. 제어흐름을 고려한 API k-gram 버스마크

3.1 소프트웨어 버스마크

소프트웨어 버스마크에 대한 정의[1]는 다음과 같다.

정의 1 (버스마크)
프로그램 *p*와 *q*에 대한 함수 *f*가 다음 조건을 만족할 때, *f(p)*를 프로그램 *p*의 버스마크라고 한다.
조건 1. *f(p)*는 부가적 정보 없이 *p* 자신에서부터 얻는다.
조건 2. 프로그램 *p*와 *q*가 서로 *copy* 관계에 있다면 *f(p)* = *f(q)* 이다.

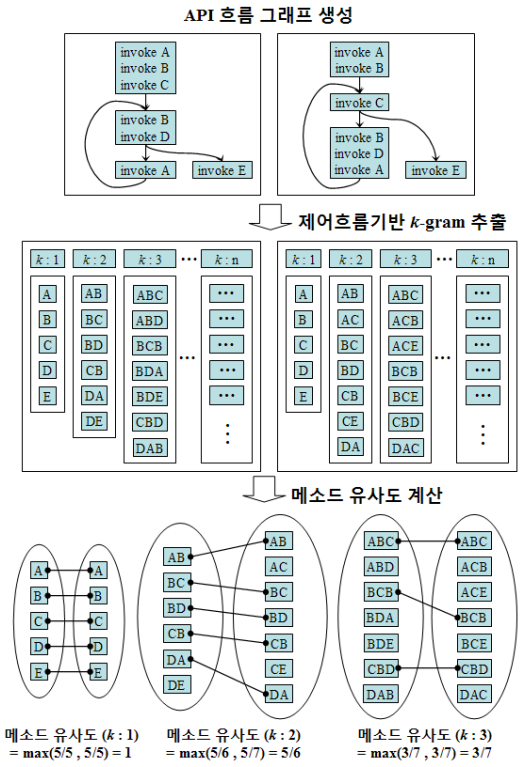
다음 두 가지 속성은 버스마크가 만족시켜야 하는 평가 기준을 나타낸다.

속성 1 (신뢰도 Credibility)
같은 기능을 하는 두 프로그램 *p*와 *q*에 대해서, *p*와 *q*가 독립적으로 개발되었을 때, *f(p)*≠ *f(q)*을 만족해야 한다.

* 이 논문은 2008년도 정부(교육과학기술부)의 재원으로 한국과학재단의 지원을 받아 수행된 연구임 (No. R01-2008-000-11856-0)

속성 2 (강인도 Resilience)
프로그램 p'이 프로그램 p로부터 변환되었다고 할 때, $f(p) = f(p')$을 만족해야 한다.

3.2 제어흐름을 고려한 API k-gram 버스마크



(그림 1) 제어흐름을 고려한 API k-gram 기반 버스마크의 추출 및 유사도 비교 방법

정의 2 (API 흐름 그래프)
메소드의 제어 흐름 그래프 $G=(V,E,Vs,Ve)$에 대해서 G의 시작 블록을 Vs, 종료 블록을 Ve라고 할 때 다음 조건을 만족하는 $G'=(V',E',Vs,Ve)$를 메소드의 API 흐름 그래프라고 정의한다.

조건 1. $V' = V - \{v \in V \mid v$에서 API를 호출하지 않는다.$\}$
조건 2. $E' = \{(vf,vt) \mid (vf,v) \in E \wedge (v,vt) \in E \wedge v \notin V'\}$

정의 3 (API k-gram)
메소드의 API 흐름 그래프 $G'=(V',E',Vs,Ve)$에 대해서 Vs에서 $e \in E'$를 따라서 Ve에 이르는 경로에서 호출되는 API k개의 시퀀스*를 API k-gram이라고 정의한다.

정의 4 (제어 흐름을 고려한 API k-gram 버스마크)
메소드로부터 생성 가능한 모든 API k-gram의 집합을 그 메소드의 제어 흐름을 고려한 API k-gram 버스마크라고 정의한다.

제어 흐름을 고려한 API k-gram 버스마크 시스템의 핵심은 (그림1)과 같다. 비교하고자 하는 클래스에 대해서 메소드 단위로 제어 흐름 그래프(이하 CFG)를 생성하고 CFG의 에지를 순회하여 정적 API k-gram 집합을 생성한다. 그리고 추출된 k-gram 집합을 비교하여 메소드 유사도를 계산하고, 메소드 유사도를 종합하여 클래스 유사도를 계산한다.

API k-gram 버스마크는 비교 대상인 두 메소드의 유사도를 구하기 위해서 2차원 구조인 메소드의 CFG를 직접비교하는 대신에 그 메소드의 CFG로부터 생성 가능한 API k-gram 집합을 이용해서 비교하는 방법을 선택한 것이다. 일반적으로 NP 복잡도를 가지는 그래프 비교 문제를 API k-gram의 비교 문제로 바꾸면 두 메소드의 API k-gram 개수가 각각 M과 N일 때, 시간 복잡도를 $O(M \cdot N)$으로 줄일 수 있다는 장점이 있다.

4. API k-gram 버스마크의 생성

4.1 바이트코드의 제어 흐름 그래프 생성

자바 클래스 파일로부터 CFG를 생성하기 위해서는 다음과 같은 과정을 거친다.

먼저 클래스 파일을 JDK에 포함되어 있는 `javap` 프로그램으로 디스어셈블하여 JVM 명령어 리스트로 변경한다. 그리고 JVM 명령어 중에서 조건 및 무조건 분기문을 해석하여 CFG를 생성한다. CFG 생성 방법은 컴파일러 이론에서 일반적으로 언급되는 알고리즘을 따른다. 즉, `ifeq`, `ifne`와 같은 조건 분기문이나 `goto`, `jsr`와 같은 무조건 분기문에서 기본 블록이 분리되고, 분기문의 목표 지점을 중심으로 기본 블록이 분리된다.

JVM 명령어로부터 CFG를 생성할 때 특별하게 고려해야 할 사항은 예외 상황(exception)에 대한 처리이다. 예외 상황 처리는 원칙적으로 예외 범위(exception range)에 포함된 JVM 명령어 중에서 예외 상황을 발생 시킬 수 있는 모든 명령어를 예외 처리 루틴(exception handler)에 연결해야 한다. 이 경우에 예외 범위에 포함된 모든 명령어들이 각각 기본 블록이 되는 상황이 발생하기 때문에 CFG의 에지가 급격히 증가하는 문제가 발생한다. 따라서 본 논문에서는 예외 범위에 포함된 기본 블록 단위로 예외 처리 루틴을 연결하는 방법을 선택함으로써 예외 상황에 대한 에지 추가의 부담을 최소화했다.

4.2 제어 흐름을 고려한 API k-gram 추출

CFG로부터 API k-gram을 추출하기 위해서는 두 단계 작업이 필요하다. 먼저 CFG를 API 흐름 그래프로 변경해 주는 작업이 선행되고, 그 다음에 API 흐름 그래프로부터 API k-gram을 추출하는 작업이 뒤따른다.

먼저 주어진 CFG로부터 API 흐름 그래프를 생성하는 방법은 다음과 같다. CFG를 구성하는 기본 블록 중에서 API 호출을 포함하지 않는 기본 블록을 삭제한다. 그와 동시에 삭제된 기본 블록으로 진입하는 에지와 삭제된 기본 블록에서 나가는 에지를 연결시키는 작업을 추가한다. 그 이유는 API 호출이 없는 기본 블록을 삭제하면서

* Object, String, PrintStream에 포함된 API와 같이 프로그램의 특성을 나타내기에 부적합한 범용 API는 추출 대상에서 제외한다.

CFG를 구성하는 에지가 끊어지는 것을 막기 위함이다.

API 흐름 그래프가 완성되면 API k-gram을 추출할 수 있다. API 흐름 그래프를 구성하는 각각의 기본 블록을 시작 지점으로 하여 k개의 API 호출 시퀀스를 수집하는 작업을 반복한다. 만일 현재 방문하고 있는 기본 블록에서 n개(n < k)의 API 호출 시퀀스를 수집했다면 제어 흐름에 의해서 인접한 노드에서는 k - n개의 API 호출을 수집하면 된다. 즉, 제어 흐름에 의해서 인접한 기본 블록에서 k - n 개의 API 호출을 수집하는 재귀적인 알고리즘이다. 알고리즘의 종료 조건은 2가지로 나눌 수 있다. k개의 API 호출 시퀀스를 수집 완료, 즉 k - n 값이 0이 되거나, k - n 값이 0이 아니더라도 말단 블록에 도달한 경우이다.

5. API k-gram 버스마크의 유사도 계산

정의 8 (메소드 유사도)

두 메소드 A와 B로부터 생성된 k-gram 집합을 각각 $kg(A)$와 $kg(B)$라고 할 때, 두 메소드 A와 B의 유사도 M_{sim}은 다음과 같이 정의한다.

$$M_{sim}(A,B) = max\left(\frac{|kg(A) \cap kg(B)|}{|kg(A)|}, \frac{|kg(A) \cap kg(B)|}{|kg(B)|}\right)$$

메소드 유사도의 의미는 원본 메소드를 구성하고 있는 k-gram을 도용 메소드가 얼마나 많이 포함하고 있는지에 대한 비율이다. 두 메소드 A와 B중에서 어느 메소드가 원본인지 알 수 없는 상황에서도 유사도를 계산하기 위해서 A와 B 기준으로 각각 유사도를 구하여 큰 값을 선택한다.

정의 9 (클래스 유사도)

두 클래스 C와 D에 포함된 메소드가 각각 $M[1..m]$과 $N[1..n]$일 때, 두 클래스의 유사도 C_{sim}은 다음과 같이 정의한다.

$$map(M[i]) = max(\ |kg(M[i]) \cap kg(N[1])|,...,|kg(M[i]) \cap kg(N[n])|\ )$$
$$map(N[j]) = max(\ |kg(M[1]) \cap kg(N[j])|,...,|kg(M[m]) \cap kg(N[j])|\ )$$

$$C_{sim}(C,D) = max\left(\frac{\sum_{i=1}^{m} map(M[i])}{\sum_{i=1}^{m} |kg(M[i])|}, \frac{\sum_{j=1}^{n} map(N[j])}{\sum_{j=1}^{n} |kg(N[j])|}\right)$$

클래스 유사도의 의미는 원본 클래스의 모든 메소드로부터 도용 클래스의 메소드에 대한 매핑을 구하는 것이다. 즉, 원본 클래스의 모든 메소드에 대해서 도용 클래스의 모든 메소드와의 API k-gram을 구하여 최대 매핑값을 모두 더하고 그 값을 원본 클래스를 구성하고 있는 API k-gram의 개수로 나눈 값이다. 두 클래스 C와 D 중에서 어느 클래스가 원본인지 알 수 없는 상황에서도 유사도를 계산하기 위해서 C와 D 기준으로 각각 유사도를 구하여 큰 값을 선택한다.

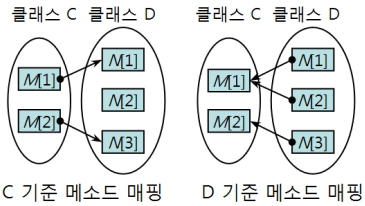
<표 1> 예제 클래스 C와 D에 포함된 모든 메소드 사이의 API k-gram 교집합 개수 계산 결과

D 클래스 / C 클래스	$N[1]$ ($\|kg(N[1])\|$=9)	$N[2]$ ($\|kg(N[2])\|$=5)	$N[3]$ ($\|kg(N[3])\|$=7)
$M[1]$ ($\|kg(M[1])\|$=8)	7	4	1
$M[2]$ ($\|kg(M[2])\|$=7)	5	3	6

표 1의 예제로 메소드 유사도와 클래스 유사도를 계산해 보면 다음과 같다. 먼저 메소드 $M[1]$과 $N[1]$의 유사도는 메소드 유사도 정의에 의해서 다음과 같이 구할 수 있다.

$$M_{sim}(M[1],N[1]) = max(7/8,\ 7/9) = 7/8$$

즉, API k-gram 개수가 8인 메소드 $M[1]$과 개수가 9인 메소드 $N[1]$으로부터 얻을 수 있는 최대 교집합 개수인 8을 기준으로 7개의 교집합을 얻었기 때문에 메소드 유사도는 7/8이다.



(그림 2) 메소드 매핑 방법 비교

클래스 유사도는 (그림 2)와 같이 클래스에 포함된 메소드를 매핑하는 방법으로 구할 수 있다. <표 1>에서 클래스 C를 기준으로 교집합의 개수가 가장 많은 메소드를 클래스 D에서 선택하면 7과 6이 선택된다. 즉, $M[1]$를 $N[1]$에 매핑시키고 $M[2]$를 $N[3]$에 매핑시켜서 최대 7+6=13개의 교집합 개수를 얻는다. 반대로 클래스 D를 기준으로 교집합의 개수가 가장 많은 클래스 C에서 선택하면 7, 4, 6이 선택된다. 즉, $N[1]$를 $M[1]$에 매핑시키고 $N[2]$를 $M[1]$에 매핑시키고, $N[3]$를 $M[2]$에 매핑시켜서 최대 교집합 개수인 7+4+6=17개를 얻는다. 이와 같은 메소드 매핑 결과는 그림 2와 같다. 매핑된 최대 교집합 개수의 합인 13과 17을 클래스에 포함된 메소드의 API k-gram의 개수의 합인 15와 21로 나눈 값 중에서 큰 값을 취한 것이 클래스 유사도이다.

$$C_{sim}(C,D) = max((7+6)/15, (7+4+6)/21)\ )$$
$$= max(\ 13/15, 17/21\ ) = 13/15$$

6. 실험 및 평가

제어 흐름을 고려한 API k-gram기반 버스마크의 효용성을 검증하기 위하여 대표적인 정적 버스마크 기법인 Tamada, k-gram 버스마크와 비교 실험을 하였다. 버스마크는 비슷한 프로그램이더라도 서로 독립적으로 구현된 프로그램을 구별할 수 있는 능력이 중요하기 때문에 오픈소스로 공개된 PNG 프로그램을 실험 대상으로 선정하였다. 실험에 사용된 프로그램은 <표2>와 같다.

<표 2> 실험 대상 프로그램과 k값 증가에 따른 API k-gram 추출이 가능한 클래스 개수의 변화

실험 대상 프로그램	클래스 개수								
	원본	***k:1***	***k:2***	***k:3***	***k:4***	***k:5***	***k:6***	***k:7***	***k:8***
JIMI 1.0	324	202	157	134	122	115	108	102	98
JIU 0.14.2	235	147	99	82	74	71	69	67	67
Sixlegs 2.0.4	41	32	21	18	15	12	11	10	9
Visualtek 1.6	12	8	7	6	6	6	6	6	6
JAI 1.1.2	472	404	347	290	229	191	176	162	157

제어 흐름을 고려한 API k-gram 버스마크에 대한 실

험은 먼저 적절한 k값의 선택하는 것이 중요하다. <표 2>에서 k값이 증가할 때 API k-gram 추출이 가능한 클래스 개수의 변화를 볼 수 있다. 예를 들어, JIMI 프로그램의 경우 포함된 클래스는 모두 324개였으나 k값이 3으로 증가하면 134개의 클래스에서만 API k-gram 추출이 가능한 것을 확인할 수 있다. k가 커질수록 서로 다른 프로그램을 구별할 수 있는 능력인 신뢰도가 높아지지만 강인성이 낮아진다. 본 논문에서는 신뢰도와 강인도를 모두 고려할 수 있는 k값으로 3을 선택하여 실험을 하였다. 즉, API 가 빈번하게 호출되지 않는 작은 클래스는 실험대상에서 제외된다. 여기서 API k-gram 버스마크의 문제점이 발생한다. Tamada 버스마크와 k-gram 버스마크는 클래스 크기와 상관없이 모든 클래스를 비교 실험 대상으로 할 수 있지만 API k-gram은 API 호출이 빈번한 클래스만을 실험 대상으로 선택해야 한다.

<표 3> 버스마크의 신뢰도 실험

		JIU	Sixlegs	Visualtek	JAI
JIMI	Tamada	0.1046	0.1052	0.1005	0.0742
	3-gram	0.0321	0.0251	0.0349	0.0498
	API 3-gram	0.0116	0.0102	0.0222	0.0033
JIU	Tamada	-	0.2561	0.2284	0.1075
	3-gram	-	0.0302	0.0332	0.0152
	API 3-gram	-	0.0068	0.0007	0.0106
Sixlegs	Tamada	-	-	0.2397	0.1312
	3-gram	-	-	0.0282	0.0195
	API 3-gram	-	-	0.0000	0.0020
Visualtek	Tamada	-	-	-	0.0780
	3-gram	-	-	-	0.0149
	API 3-gram	-	-	-	0.0002

*** Tamada 유사도 평균 : 0.1425**

*** 3-gram 유사도 평균 : 0.0283**

*** API 3-gram 유사도 평균 : 0.0068**

신뢰도 실험을 위해서 서로 다른 두 프로그램에 포함되어 있는 모든 클래스의 조합에 대해서 각각 유사도를 계산한 다음 평균값을 구하였다. <표 3>은 버스마크의 신뢰도 실험 결과이다. 신뢰도 실험에서는 유사도가 낮을수록 신뢰도가 높다. 즉, 유사도가 낮을수록 서로 다른 프로그램을 구별하는 능력이 높다는 것을 의미한다. 실험 결과 Tamada, k-gram 버스마크와 비교하여 API k-gram 버스마크가 가장 신뢰도가 높았다.

Tamada 버스마크와 k-gram 버스마크의 신뢰도가 API k-gram 버스마크보다 떨어지는 이유는, 충분히 특성화된 요소를 대상으로 버스마크를 추출하지 않았기 때문이다. 즉, 일반적인 클래스들이 가질 수 있는 범용적인 요소들에 의해서 유사도 평균이 올라가는 문제가 생긴다. 그러나 API k-gram 버스마크는 범용적이지 않고 대체하기 어려운 자바 표준 API 호출 시퀀스로 유사도를 비교하기 때문에 높은 신뢰도 값을 얻을 수 있었다.

<표 4>는 난독화 도구인 Smokescreen[4]과 Jikes 컴파일러[5]를 이용한 강인도 실험 결과이다. 이 실험에서 API k-gram 버스마크는 Tamada, k-gram 버스마크와 비교하여 높은 강인도를 보였다.

<표 4> 버스마크의 강인도 실험

	Smokescreen			Jikes		
	Tamada	3-gram	API 3-gram	Tamada	3-gram	API 3-gram
JIMI	0.7533	0.6661	1.0000	0.7881	0.8782	1.0000
JIU	0.6923	0.6431	1.0000	0.8509	0.8732	0.9993
Sixlegs	0.6578	0.5963	1.0000	0.8855	0.8753	1.0000
Visualtek	0.6861	0.6451	1.0000	0.7503	0.8392	1.0000
JAI	0.6821	0.6216	0.9999	0.7773	0.8595	0.9999
평균	0.6943	0.6344	1.0000	0.8104	0.8651	0.9998

난독화 도구는 메소드 순서를 변경하거나 스택 연산의 순서를 바꾸는 변환을 시행하는데 Tamada 버스마크는 메소드 호출 순서를 추출할 때 물리적인 위치 정보를 이용해서 추출하기 때문에 메소드 순서 변경에 대해서 취약하다. k-gram 버스마크도 JVM 명령어의 시퀀스를 추출할 때 제어 흐름을 고려하지 않기 때문에 난독화에 취약하다. Jikes 컴파일러는 Javac 컴파일러가 ifgt로 컴파일했던 조건문을 ifle로 바꾸기도 한다. 이때 True, False 블록의 위치가 바뀌고 JVM 명령어도 일부 변경되는데 Tamada, k-gram은 이런 변환에도 취약하다.

그러나 API k-gram 버스마크는 다른 것으로 대체하기 어려운 API를 추출하고 또한 제어 흐름까지 고려하기 때문에 메소드 위치 변경이나 뒤바뀐 분기문에 상관없이 항상 높은 강인도를 얻을 수 있었다.

7. 결론 및 향후 연구 과제

본 논문에서는 자바 메소드의 제어 흐름을 고려한 API k-gram을 추출하여 버스마크로 사용하는 방법을 제안하였다. 본 논문에서 제안한 버스마크에 대한 효용성을 평가하기 위해서 신뢰도와 강인도 실험을 하였고, 실험 결과로부터 API k-gram 버스마크가 기존의 Tamada, k-gram 버스마크와 비교하여 신뢰도와 강인도가 모두 높다는 것을 확인하였다.

그러나 API k-gram 버스마크는 API 호출이 빈번하지 않은 작은 클래스의 경우에는 비교 자체를 할 수 없다는 문제가 있다. 이것을 보완하기 위한 향후 연구로써 기존의 API 호출에 덧붙여서 자바의 고유한 특성을 표현하는 다른 요소를 추가하는 방법을 고려하고 있다. 또한 함수간 분석(Interprocedural Analysis)을 통해서 작은 API k-gram 조각을 연결시키는 방법도 고려하고 있다.

참고 문헌

[1] Tamada, H., Nakamura, M., Monden, A., Matsumoto, K. Java birthmarks - Detecting the software theft. IEICE Transactions on Information and Systems, Vol. E88-D, Issue 9, pp. 2148-2158, 2005.

[2] Ginger Myles and Christian Collberg. k-gram Based Software Birthmarks. In Proceeding of the 2005 ACM Symposium on Applied Computing, pp. 314-318. Santa Fe, New Mexico, USA, 2005.

[3] 박희완, 최석우, 임현일, 한태숙. 정적 API 트레이스 버스마크를 이용한 자바 클래스 도용 탐지. 한국정보과학회 컴퓨터 종합학술대회 2008, Vol.35, No.1, pp. 287-288, 2008.

[4] "Smokescreen" http://www.leesw.com/smokescreen/

[5] "Jikes Java Compiler" http://jikes.sourceforge.net/