(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2022-0037721
(43) 공개일자 2022년03월25일

(51) 국제특허분류(Int. Cl.)
G06F 8/41 (2018.01) *G06F 21/12* (2013.01)
G06F 8/38 (2018.01)
(52) CPC특허분류
G06F 8/443 (2013.01)
G06F 21/125 (2013.01)
(21) 출원번호 10-2020-0120537
(22) 출원일자 2020년09월18일
심사청구일자 2020년09월18일

(71) 출원인
국방과학연구소
대전광역시 유성구 북유성대로488번길 160 (수남동)
고려대학교 산학협력단
서울특별시 성북구 안암로 145, 고려대학교 (안암동5가)
(72) 발명자
이영비
서울특별시 성북구 안암로 145 로봇융합관 403호
석재혁
서울특별시 성북구 안암로 145 로봇융합관 403호
(뒷면에 계속)
(74) 대리인
특허법인 광장리앤고

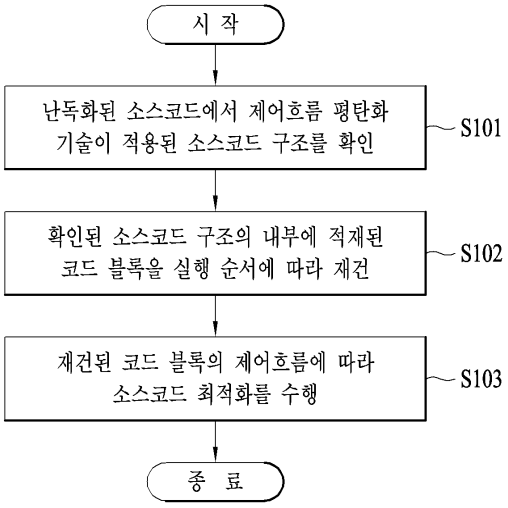
전체 청구항 수 : 총 10 항

(54) 발명의 명칭 **소스코드 재건 및 최적화 방법 및 이를 위한 장치**

(57) 요 약

본 발명은 소스코드 재건 및 최적화 방법으로서, 난독화 기술이 적용된 소스코드 구조를 확인하는 단계; 확인된 소스코드 구조의 내부에 적재된 코드 블록을 실행 순서에 따라 재건하는 단계; 및 재건된 코드 블록의 제어흐름에 따라 소스코드 최적화를 수행하는 단계를 포함하고, 이를 위한 소스코드 재건 및 최적화 장치를 제공한다.

대 표 도 - 도1



(52) CPC특허분류
G06F 8/38 (2013.01)
G06F 8/423 (2013.01)
G06F 8/433 (2013.01)
(72) 발명자
이재융
서울특별시 서초구 서초대로38길 12
이동훈
서울특별시 성북구 안암로 145 로봇융합관 403호
김현숙
대전광역시 유성구 북유성대로488번길 160
신임섭
대전광역시 유성구 북유성대로488번길 160

명 세 서

청구범위

청구항 1

소스코드 재건 및 최적화 방법에 있어서,

난독화 기술이 적용된 소스코드 구조를 확인하는 단계;

상기 확인된 소스코드 구조의 내부에 적재된 코드 블록을 실행 순서에 따라 재건(reconstruction)하는 단계; 및

상기 재건된 코드 블록의 제어흐름에 따라 소스코드 최적화(optimization)를 수행하는 단계를 포함하는, 방법.

청구항 2

제1항에 있어서,

상기 재건하는 단계는,

상기 적재된 코드 블록의 각각에서 레이블(label)의 위치를 확인하는 단계;

상기 레이블의 아래에 코드 블록 인덱스를 출력하기 위한 인터페이스를 삽입하는 단계; 및

상기 인터페이스가 삽입된 코드 블록을 실행하여 상기 코드 블록 인덱스를 출력하는 단계를 포함하는, 방법.

청구항 3

제2항에 있어서,

상기 출력하는 단계에서, 상기 코드 블록 인덱스는, 상기 적재된 코드 블록 중 실제 실행되는 코드 블록에 대해 실행 순서대로 출력되는, 방법.

청구항 4

제3항에 있어서,

상기 출력된 코드 블록 인덱스에 대응하는 코드 블록을 상기 실행 순서대로 배치하는 단계를 더 포함하는, 방법.

청구항 5

제4항에 있어서,

상기 배치된 코드 블록은 상기 레이블과 코드 블록 이동 명령문이 블록 내에서 삭제되어 배치되는, 방법.

청구항 6

제4항에 있어서,

상기 배치된 코드 블록에서 디스패처(dispatcher) 블록을 제거하는 단계를 더 포함하고,

상기 디스패처 블록은 상기 적재된 코드 블록 중 실행할 코드 블록을 선택하기 위한 선택 블록인, 방법.

청구항 7

제1항에 있어서,

상기 소스코드 최적화를 수행하는 단계는,

더 이상 제거 및 최적화 할 소스코드가 없을 때까지 상기 재건된 코드 블록의 소스코드를 제거 및 최적화를 반복하는 단계를 포함하는, 방법.

청구항 8

제1항에 있어서,

상기 최적화가 수행된 소스코드에 대해 분석 및 컴파일을 수행하는 단계를 더 포함하는, 방법.

청구항 9

소스코드 재건 및 최적화 장치에 있어서,

추출부를 통해, 난독화 기술이 적용된 소스코드 구조를 확인하고,

재건부를 통해, 상기 확인된 소스코드 구조의 내부에 적재된 코드 블록을 실행 순서에 따라 재건하고,

최적화부를 통해, 상기 재건된 코드 블록의 제어흐름에 따라 소스코드 최적화를 수행하는, 장치.

청구항 10

소스코드 재건 및 최적화 방법을 컴퓨터에서 실행시키기 위한 프로그램을 기록한 컴퓨터로 읽을 수 있는 비일시적 기록매체로서,

상기 소스코드 재건 및 최적화 방법은,

난독화 기술이 적용된 소스코드 구조를 확인하는 단계;

상기 확인된 소스코드 구조의 내부에 적재된 코드 블록을 실행 순서에 따라 재건하는 단계; 및

상기 재건된 코드 블록의 제어흐름에 따라 소스코드 최적화를 수행하는 단계를 포함하는, 비일시적 기록매체.

발명의 설명

기 술 분 야

[0001] 본 발명은 소스코드 재건 및 최적화 방법 및 이를 위한 장치에 관한 발명으로, 보다 구체적으로는, 소스코드 레벨에서 제어흐름 난독화 기술이 적용된 경우, 이를 컴파일하기 전에 재건 및 최적화를 수행하는 소스코드 재건 및 최적화 방법 및 장치에 관한 것이다.

배 경 기 술

[0002] 난독화 기술은 소프트웨어의 지적재산권을 보호하기 위하여 소프트웨어의 기능성은 그대로 유지하면서 자료구조, 제어흐름 등을 변형하는 기법이다. 이러한 난독화 기술은 컴파일이 수행되기 전인 소스코드 레벨에 적용될 수도 있고 컴파일이 수행된 후인 바이너리 레벨에 적용될 수도 있다. 컴파일러에서 컴파일을 수행하는 과정에서 소스코드에 최적화 기법을 적용하게 되는데, 해당 최적화 기법이 적용되면 난독 변환된 부분이 제거된다. 따라서, 소스코드 레벨에서 난독화가 이루어지는 경우 난독화 기술에 의한 보호 효과가 크게 감소할 수 있다. 이 때문에, 일반적으로 컴파일이 수행된 후인 바이너리 레벨에 난독화 기술을 적용하는 것이 선호된다. 따라서, 소프트웨어 제작사에서는 자사에서 제작한 소프트웨어를 보호하기 위하여 난독화 기술을 적용할 때, 소스코드를 제공하여야 하는 경우가 아니라면 바이너리 레벨에서 난독화 기술을 적용하여 배포하였다. 그리고 이러한 소스코드 레벨에서 난독화의 비선호도는 소스코드 레벨에서의 난독화 기술이 많이 연구되지 않는 이유 중 하나이다.

[0003] 난독화 기술은 구획 난독화(변수명/함수명 변경, 주석 제거 등), 데이터 난독화(변수값 변형, 자료구조 변형 등), 제어흐름 난독화(제어흐름 변형, 불필요한 코드 삽입 등) 및 방지 난독화(디버깅 탐지, 가상머신 탐지 등)의 4개 범주로 분류될 수 있다. 종래의 소스코드 레벨에서의 상용 난독화 도구는 주로 변수명/함수명을 변경하거나, 소스코드 내의 주석을 제거하거나, 변수값을 변형하는 수준이었다. 즉, 소스코드 레벨에서는 배포가 목적인 경우가 대부분이어서, 주로 구획 난독화와 데이터 난독화 기술이 적용되었다. 바이너리 레벨에서라면 구획 난독화, 데이터 난독화, 제어흐름 난독화, 그리고 방지 난독화 기술을 모두 적용할 수 있지만, 소스코드 레벨에서는 상세한 주소 정보나 어셈블리어를 가지고 있지 않기 때문에 설계/구현하기 힘든 난독화 기술이 있었다.

[0004] 다만, 구획 난독화와 데이터 난독화는 앞서 서술한 것처럼 컴파일러에서 컴파일하는 과정에서 모두 최적화되어 무력화되는 타입의 난독화 기술이다. 따라서 소스코드 레벨에서는 주로 구획 난독화와 데이터 난독화 기술이 적

용되므로, 소스코드 레벨에서의 난독화 기술을 역난독화하는 기술로써 최적화 기술이 활용되었다.

[0005] 한편, 최근 연구에서는 소스코드 레벨에서 적용 가능한 제어흐름 난독화 기술이 설계 및 구현되어 상용 난독화 도구에 탑재되고 있다. 제어흐름 난독화 기술은 앞선 구획 난독화와 데이터 난독화 기술과는 달리 컴파일러 내부의 최적화기로는 최적화되지 않는다. 이는 제어흐름 난독화 기술이 소스코드의 내부 구조를 강하게 변형시키는데, 해당 변형부를 먼저 재건하여야 최적화를 통한 가독성의 증가를 기대할 수 있다는 것을 의미한다. 따라서 소스코드 레벨에서의 제어흐름 난독화 기술에 대응하기 위해 재건 기술에 대해 연구가 이루어지고 있다.

발명의 내용

해결하려는 과제

[0006] 본 실시 예가 해결하고자 하는 과제는, 제어흐름 난독화 기술이 적용된 소스코드에 대해 소스코드 레벨에서 제어흐름의 재건을 선행한 후 최적화하여, 컴파일 과정에서 컴파일러 최적화 옵션을 전혀 적용하지 않더라도 매우 높은 최적화율과 가독성을 제공하는 소스코드 재건 및 최적화 방법 및 장치를 제공하는 데 있다.

[0007] 본 실시 예가 이루고자 하는 기술적 과제는 상기된 바와 같은 기술적 과제들로 한정되지 않으며, 이하의 실시 예들로부터 또 다른 기술적 과제들이 유추될 수 있다.

과제의 해결 수단

[0008] 일 실시 예에 따른 소스코드 재건 및 최적화 방법은, 제어흐름 난독화 기술이 적용된 소스코드 구조를 확인하는 단계; 상기 확인된 소스코드 구조의 내부에 적재된 코드 블록을 실행 순서에 따라 재건(reconstruction)하는 단계; 및 상기 재건된 코드 블록의 제어흐름에 따라 소스코드 최적화(optimization)를 수행하는 단계를 포함할 수 있다.

[0009] 일 실시 예에 따른 소스코드 재건 및 최적화 장치는, 추출부를 통해, 제어흐름 난독화 기술이 적용된 소스코드 구조를 확인하고, 재건부를 통해, 상기 확인된 소스코드 구조의 내부에 적재된 코드 블록을 실행 순서에 따라 재건하고, 최적화부를 통해, 상기 재건된 코드 블록의 제어흐름에 따라 소스코드 최적화를 수행할 수 있다.

[0010] 일 실시 예에 따른 소스코드 재건 및 최적화 방법을 컴퓨터에서 실행시키기 위한 프로그램을 기록한 컴퓨터로 읽을 수 있는 비일시적 기록매체로서, 상기 소스코드 재건 및 최적화 방법은, 제어흐름 난독화 기술이 적용된 소스코드 구조를 확인하는 단계; 상기 확인된 소스코드 구조의 내부에 적재된 코드 블록을 실행 순서에 따라 재건하는 단계; 및 상기 재건된 코드 블록의 제어흐름에 따라 소스코드 최적화를 수행하는 단계를 포함할 수 있다.

발명의 효과

[0011] 본 개시에 따르면, 소스코드 재건 및 최적화 방법은 컴파일러 최적화기에서 적용할 수 없는 동적 분석의 형태를 적용하고 있기 때문에 더욱 정확한 형태의 제어흐름 재건이 가능하다.

[0012] 또한, 본 개시의 소스코드 재건 및 최적화 방법은 제어흐름 재건을 수행함으로써 이로 인해 최적화 가능 패턴이 추가 생성되므로, 컴파일 과정에서 컴파일러 최적화 옵션을 전혀 적용하지 않더라도 매우 높은 최적화율과 가독성을 제공할 수 있다.

[0013] 또한, 본 개시의 소스코드 재건 및 최적화 방법은 소스코드간 변환(Source-to-Source Transformation)이기 때문에 일반적인 코드 분석가에게 다소 친숙하며, 컴파일러의 최적화 기술을 적용하였을 때 보다 생성 결과물의 가독성이 높다.

[0014] 발명의 효과는 이상에서 언급한 효과로 제한되지 않으며, 언급되지 않은 또 다른 효과들은 청구범위의 기재로부터 당해 기술 분야의 통상의 기술자에게 명확하게 이해될 수 있을 것이다.

도면의 간단한 설명

[0015] 도 1은 일 실시 예에 따른 소스코드 재건 및 최적화 방법의 흐름도이다.

도 2a는 제어흐름 평탄화 기술이 적용되기 전의 원본 소스코드이다.

도 2b는 제어흐름 평탄화 기술이 적용되기 전의 제어흐름 그래프이다.

도 2c는 제어흐름 평탄화 기술이 적용되기 전의 소스코드 실행 순서이다.

도 3a는 제어흐름 평탄화 기술이 적용된 소스코드이다.

도 3b는 제어흐름 평탄화 기술이 적용된 제어흐름 그래프이다.

도 3c는 제어흐름 평탄화 기술이 적용된 소스코드 실행 순서이다.

도 4는 일 실시 예에 따른 소스코드 재건 및 최적화 방법의 구조도이다.

도 5는 일 실시 예에 따른 소스코드 재건 및 최적화 방법의 알고리즘의 흐름도이다.

도 6은 일 실시 예에 따른 제어흐름 재건을 나타낸 개념도이다.

도 7은 일 실시 예에 따른 소스코드 재건 및 최적화 장치의 블록도이다.

발명을 실시하기 위한 구체적인 내용

[0016] 실시 예들에서 사용되는 용어는 본 개시에서의 기능을 고려하면서 가능한 현재 널리 사용되는 일반적인 용어들을 선택하였으나, 이는 당 분야에 종사하는 기술자의 의도 또는 판례, 새로운 기술의 출현 등에 따라 달라질 수 있다. 또한, 특정한 경우는 출원인이 임의로 선정한 용어도 있으며, 이 경우 해당되는 설명 부분에서 상세히 그 의미를 기재할 것이다. 따라서 본 개시에서 사용되는 용어는 단순한 용어의 명칭이 아닌, 그 용어가 가지는 의미와 본 개시의 전반에 걸친 내용을 토대로 정의되어야 한다.

[0017] 명세서 전체에서 어떤 부분이 어떤 구성요소를 "포함"한다고 할 때, 이는 특별히 반대되는 기재가 없는 한 다른 구성요소를 제외하는 것이 아니라 다른 구성요소를 더 포함할 수 있음을 의미한다. 또한, 명세서에 기재된 "...부", "...모듈" 등의 용어는 적어도 하나의 기능이나 동작을 처리하는 단위를 의미하며, 이는 하드웨어 또는 소프트웨어로 구현되거나 하드웨어와 소프트웨어의 결합으로 구현될 수 있다.

[0018] 명세서 전체에서 기재된 "a, b, 및 c 중 적어도 하나"의 표현은, 'a 단독', 'b 단독', 'c 단독', 'a 및 b', 'a 및 c', 'b 및 c', 또는 'a, b, 및 c 모두'를 포괄할 수 있다.

[0019] 이하에서 언급되는 "단말"은 네트워크를 통해 서버나 타 단말에 접속할 수 있는 컴퓨터나 휴대용 단말로 구현될 수 있다. 여기서, 컴퓨터는 예를 들어, 웹 브라우저(WEB Browser)가 탑재된 노트북, 데스크톱(desktop), 랩톱(laptop) 등을 포함하고, 휴대용 단말은 예를 들어, 휴대성과 이동성이 보장되는 무선 통신 장치로서, IMT(International Mobile Telecommunication), CDMA(Code Division Multiple Access), W-CDMA(W-Code Division Multiple Access), LTE(Long Term Evolution) 등의 통신 기반 단말, 스마트폰, 태블릿 PC 등과 같은 모든 종류의 핸드헬드(Handheld) 기반의 무선 통신 장치를 포함할 수 있다.

[0020] 아래에서는 첨부한 도면을 참고하여 본 개시의 실시 예에 대하여 본 개시가 속하는 기술 분야에서 통상의 지식을 가진 자가 용이하게 실시할 수 있도록 상세히 설명한다. 그러나 본 개시는 여러 가지 상이한 형태로 구현될 수 있으며 여기에서 설명하는 실시 예에 한정되지 않는다. 이하에서는 도면을 참조하여 본 개시의 실시 예들을 상세히 설명한다.

[0021] 도 1은 일 실시 예에 따른 소스코드 재건 및 최적화 방법의 흐름도이다.

[0022] 단계 S101에서 난독화된 소스코드에서 제어흐름 평탄화 기술이 적용된 소스코드 구조를 확인할 수 있다. 이 때의 난독화된 소스코드는 제어흐름 난독화 기술이 적용된 소스코드이고, 제어흐름 난독화 기술은 제어흐름 평탄화(Control-Flow Flattening) 기술일 수 있다. 이 기술은 일반적으로 수직 관계인 소프트웨어의 제어흐름을 수평 관계로 변형하는 기술로서, 보다 상세한 설명은 도 2 및 3을 참조로 설명하기로 한다. 일 실시 예에 따르면, 제어흐름 평탄화 기술이 적용된 소스코드 구조를 확인하는 것은 난독화된 소스코드 내부를 분석하여 재건 및 최적화 대상이 되는 평탄화 구조의 위치를 추출하는 것을 의미할 수 있다.

[0023] 단계 S102에서는, S101에서 확인된 소스코드 구조의 내부에 적재된 코드 블록을 실행 순서에 따라 재건할 수 있다. 일 실시 예에 따르면, 코드 블록을 재건하는 것은, 적재된 코드 블록의 각각에서 레이블의 위치를 확인하고, 레이블의 하단에 코드 블록 인덱스를 출력하기 위한 인터페이스를 삽입하여 실행함으로써 코드 블록 인덱스를 출력하는 과정을 포함할 수 있다. 이 때 코드 블록 인덱스는 적재된 코드 블록 중 실제 실행되는 코드 블록에 대해 실행 순서대로 출력될 수 있다. 일 실시 예에 따르면, 출력된 코드 블록 인덱스에 대응하는 코드 블록을 레이블과 코드 블록 이동 명령문을 블록 내에서 삭제한 채로 실행 순서대로 배치할 수 있다. 또한, 배치

된 코드 블록에서 디스패처(dispatcher) 블록을 제거하여 제어흐름 재건을 완료할 수 있다.

[0024] 단계 S103에서는, S102에서 재건된 코드 블록의 제어흐름에 따라 소스코드 최적화를 수행할 수 있다. 일 실시예에 따르면, 더 이상 제거 및 최적화 할 소스코드가 없을 때까지 재건된 코드 블록의 소스코드를 제거 및 최적화하는 단계를 반복할 수 있다.

[0025] 일 실시 예에 따르면, 최적화 된 소스코드에 대하여 분석을 수행할 수 있다. 분석을 통해 해당 소스코드가 어느 정도 가독성이 있는지, 난독화된 소스코드 대비 얼만큼 최적화가 수행되었는지 알 수 있다. 그리고 이러한 최적화가 완료된 소스코드를 컴파일러를 통해 컴파일하여 배포할 수 있다.

[0026] 한편, 소스코드 레벨에 적용 가능한 상용 난독화 도구는 그 수가 많지 않으며, 그 중에서도 제어흐름 난독화 기술을 적용할 수 있는 상용 난독화 도구는 더욱 적다. 제어흐름 평탄화 기술은 소스코드 레벨에서 적용할 수 있는 제어흐름 난독화 기술 중 거의 유일한 상용화 기술에 해당한다. 따라서 제어흐름 난독화 기술을 적용할 수 있는 소스코드 레벨 상용 난독화 도구는 대부분 제어흐름 평탄화 기술을 옵션으로 제공하고 있다. 이러한 제어흐름 평탄화 기술에 대해 다음 도 2 및 3을 참조하여 설명하도록 한다.

[0027] 먼저, 도 2a는 제어흐름 평탄화 기술이 적용되기 전의 원본 소스코드이고, 도 2b는 제어흐름 평탄화 기술이 적용되기 전의 제어흐름 그래프이며, 도 2c는 제어흐름 평탄화 기술이 적용되기 전의 소스코드 실행 순서이다.

[0028] 제어흐름 평탄화 기술이 적용되기 전의 제어흐름은 수직관계인 것이 일반적이다. 도 2a의 원본 소스코드는 예시적으로 작성된 비교적 간단한 소스코드로서, "Hello"와 "World"를 출력하기 위한 소스코드이다. 이러한 수직적인 구조의 소스코드는, 도 2b를 참조하면, 한 개의 노드(node)로 이루어진 제어흐름을 가진다. 따라서, 해당 소스코드가 컴파일 될 때에 "Hello" "World" Exit(즉, 코드블록 탈출)로 이루어진 한 개의 코드블록 내부에서 순차적으로 실행이 되게 된다.

[0029] 소스코드 레벨에서 적용되는 제어흐름 난독화 기술로서 제어흐름 평탄화 기술이 있으며, 소스코드에 이 기술이 적용되면 제어흐름은 수평관계로 변형된다. 이와 관련하여, 도 3a는 제어흐름 평탄화 기술이 적용된 소스코드이고, 도 3b는 제어흐름 평탄화 기술이 적용된 제어흐름 그래프이며, 도 3c는 제어흐름 평탄화 기술이 적용된 소스코드 실행 순서이다.

[0030] 먼저, 제어흐름 평탄화 기술이 적용된 소스코드의 구성요소에는 디스패처(dispatcher), 디스패처 변수, 코드 블록, 코드 블록 인덱스 및 레이블(label)이 포함될 수 있다.

[0031] 디스패처는 컴파일 실행 될 코드 블록을 선택하는 선택기이다. 예를 들어, 도 3a에서 Switch-Case 문이 디스패처에 해당할 수 있다. 디스패처 변수는 컴파일 실행될 코드 블록을 선택하기 위해 사용되는 변수이다. 예를 들어, 도 3a에서 SwVar이 디스패처 변수에 해당할 수 있다.

[0032] 코드 블록은 기능성 단위로 분할된 블록을 나타내며, 기본 블록과 유사한 개념으로 볼 수 있다. 예를 들어, 도 3c에서 Label_#k 에서 Goto Switch문 까지를 k번째 코드 블록으로 지칭할 수 있다. 코드 블록 인덱스는 코드 블록의 숫자 (또는 이름)을 나타낸다. 예를 들어, 도 3c에서의 #k에 해당하며, 본 실시 예에서의 k는 1, 2, 3이다. 레이블은 코드 블록 내에서 이동 대상 주소로써 활용될 수 있다. 예를 들어, Goto Label_#2문은 인덱스가 2인 코드 블록으로 이동할 것을 나타낼 수 있다. 레이블은 도 3c에서의 Label_#k에 해당한다.

[0033] 이를 기초로 도 3을 다시 살펴보면, 제어흐름 평탄화 기술이 적용된 제어흐름은 여러 개의 코드블록으로 이루어진 수평관계로 변형된다. 도 3a는 도 2a의 원본 소스코드에 제어흐름 평탄화 기술이 적용된 소스코드로서, "Hello"와 "World"를 출력하기 위한 소스코드가 switch case 0와 switch case 1로 두개의 코드 블록으로 나누어 진 것을 확인할 수 있다. 즉, 디스패처의 실행을 선두로 하여 디스패처 변수에 따라 실행되는 코드 블록이 정해진다. 해당 코드 블록의 실행이 완료되는 시점에서 디스패처 변수는 다른 값으로 변경되며, 이는 다음에 실행될 코드 블록 인덱스를 계산하는 과정일 것이다.

[0034] 이러한 수평적인 구조의 소스코드는 도 3b를 참조하면, 복수 개의 노드로 이루어진 제어흐름을 가진다. 따라서, 해당 소스코드가 컴파일 될 때에 디스패처인 Switch-Case 블록(301)을 기준으로 선택 가능한 코드 블록 3가지, "World"를 출력하기 위한 블록(302), "Hello"를 출력하기 위한 블록(303) 및 while문을 탈출하기 위한 블록(304)이 나열되게 된다. 도 3c을 참조하면, 디스패처를 제외하고 나머지 블록(302, 303, 304)이 수평으로 나열된 것을 확인할 수 있다. 이는 수평구조로 변형된 소스코드로는 실제 컴파일하지 않는 이상 코드 블록의 실행 순서를 알 수 없음을 의미한다. 본 실시 예는 비교적 간단한 소스코드이나, 실제 사용되는 소스코드의 구조는 매우 복잡할 것이므로, 제어흐름 평탄화 기술이 적용된 소스코드는 실제로 실행해 보지 않는 이상 실제 실행되

는 코드 블록과 그 실행 순서를 파악하기 어렵다. 다시 말해, 제어흐름 평탄화 기술이 적용된 소스코드는 코드 블록의 실행 순서를 정적으로 파악하기 어렵다.

[0035] 도 3a에서는 코드 블록 인덱스가 디스패처 변수와 동일한 값(둘 다 0 또는 1의 값만을 가짐)이지만, 난독 변환 수준이 높은 상용 난독화 도구에서는 코드 블록 인덱스와 디스패처 변수 간의 상관관계를 파악할 수 없게 구성된다. 또한, 도 3의 실시 예는 디스패처 변수의 초기값(int SwVar = 0)을 쉽게 획득할 수 있으며, 이의 연산 과정(SwVar = 1) 또한 복잡하지 않기 때문에 코드 블록의 실행 순서를 정적으로도 쉽게 파악 가능하지만, 상용 난독화 도구에서는 초기값과 연산 과정 또한 정적으로 파악하기 어렵게 구성된다.

[0036] 그리고, 고수준의 제어흐름 평탄화 기술에서는 분석을 더욱 복잡하게 하기 위하여 실제 사용되지 않는 코드 블록과 코드 블록 인덱스도 다수 삽입된다. 예를 들어, 어떠한 코드 블록 내 디스패처 변수도 특정 코드 블록 인덱스를 나타내지 않으면 해당 코드 블록 인덱스가 출력되지 않고 코드 블록이 실행되지 않을 수 있다. 또는, 코드 블록 내 인덱스의 연산이 블록 순차적으로 a=a+1; a=a-1; 로 작성되는 경우와 같이 실행되어도 동일한 기능성을 유지하는 코드 블록이 있을 수 있다. 이 때문에 분석가의 입장에서는 특정 코드 블록의 실제 실행 여부를 파악하는 것에도 디스패처 변수의 분석을 선행하여야 하는 문제가 있다. 따라서, 고수준의 제어흐름 평탄화 기술이 적용된 소스코드를 분석하기 위해서는 동적 분석을 통한 코드 블록의 실행 순서 추출 과정이 선행되어야 한다. 이 과정은 컴파일러 최적화기에서는 수행할 수 없는 과정이기 때문에, 기존 컴파일러 최적화기만으로는 제어흐름 난독화가 적용된 소스코드의 가독성을 증가시키기 어렵게 된다.

[0037] 따라서, 본 발명에서 제어흐름 난독화가 적용된 소스코드의 가독성을 증가시키기 위해 최적화 이전에 소스코드 레벨에서 구조를 재건하는 방식이 제안된다.

[0038] 도 4는 일 실시 예에 따른 소스코드 재건 및 최적화 방법의 구조도이고, 도 5는 일 실시 예에 따른 소스코드 재건 및 최적화 방법의 알고리즘의 흐름도이다.

[0039] 도 4 및 5를 참조하면, 난독화된 소스코드는 본 발명에 따른 소스코드 재건 및 최적화 장치(400)에 의해 최적화된 소스코드로 변환될 수 있다. 우선, 단계 S501에서 난독화된 소스코드 파싱 모듈(Obfuscated Source Code Parsing Module)(410)을 통해 난독화된 소스코드의 내부를 판독하고 분석할 수 있다. 그리고 단계 S502에서 정규 표현식 모듈(Regular Expression Module)(420)을 통해, 소스코드 내부에서 제어흐름 평탄화 기술이 적용된 평탄화 구조의 위치를 식별하여 추출할 수 있다. 예를 들어, 개발자는 소스코드에 마커를 삽입하여 어떤 함수에 난독화 기술을 적용할지 표시할 수 있다. 이 경우 마커의 삽입으로 인하여 함수 명이 변경되게 되며, 그 함수 내에 위치한 SwitchCase 문이 일반적으로 평탄화 구조에 해당한다.

[0040] 소스코드 재건 알고리즘은 평탄화 구조를 식별하고 재건하기 위해 최적화 과정 이전에 한번 수행되므로, 알고리즘의 라운드(round)를 확인하여 수행하게 된다. 라운드는 초기 값 0으로 설정되며, 이에 따라 소스코드 재건을 거치지 않은 소스코드는 라운드가 증가하지 않고 계속 0인 상태이다. 단계 S503에서 이러한 라운드가 0인지 체크할 수 있다. 즉, 단계 S503에서 대상이 되는 소스코드 구조가 재건된 소스코드인지 확인할 수 있다.

[0041] 단계 S503에서 "예"인 경우, 즉, 재건된 적이 없는 소스코드인 경우, 단계 S504에서 레이블 추출 모듈(Label Extraction Module)(430)을 통해, 평탄화 구조에 적재된 코드 블록에 레이블을 출력하기 위한 인터페이스를 각각 삽입하고 해당 코드 블록을 실행할 수 있다. 이 때의 인터페이스는 API(Application Programming Interface)를 의미한다. 레이블 추출 모듈(430)은 평탄화 구조 내부에 적재된 코드 블록을 실행 순서대로 추출할 수 있는 모듈로서, 동적 분석을 적용하여 평탄화 구조 내부의 레이블의 위치를 파악하고, 레이블의 아래에 코드 블록 인덱스를 출력하기 위한 인터페이스를 삽입할 수 있다. 이 때 레이블의 위치를 확인하기 위해 정규 표현식 모듈(420)이 같이 사용될 수 있다.

[0042] 일 실시 예에 따르면, 코드 블록 인덱스를 출력하기 위한 API는 Print("#3")과 같은 Print문일 수 있다. 코드 블록에 레이블을 출력하기 위한 인터페이스를 삽입하고 이를 컴파일 후 실행 한다면, 실제로 실행되는 코드 블록에 대한 코드 블록 인덱스만 코드 블록의 실행 순서대로 출력되게 된다. 예를 들어, 도 3c의 예시에서 코드 블록 인덱스가 4 및 5인 코드 블록이 존재하고, 이러한 코드 블록들이 실제로는 실행되지 않는 코드 블록인 경우를 가정한다. 이 경우, 상기 API가 삽입된 평탄화 구조가 실행되면, #2 Hello #1 World #3이 그 실행 순서에 따라 출력되고, 실행되지 않는 코드 블록의 인덱스인 #4와 #5는 출력되지 않을 것이다.

[0043] 코드 블록 인덱스가 적재된 코드 블록 중 실제 실행되는 코드 블록에 대해 실행 순서대로 출력된 이후 다음의 단계 S505 내지 S508의 소스코드 재건 과정을 거치게 된다. 먼저, 단계 S505에서 제어흐름 재건 모듈(Control-Flow Reconstruction Module)(440)을 통해, 브랜치 명령문을 재건할 수 있다. 다시 말해, 단계 S504에서 출력

된 코드 블록 인덱스에 대응하는 코드 블록을 실행 순서대로 배치할 수 있다.

[0044] 그리고 단계 S506에서, 실행 순서대로 배치된 코드 블록의 레이블을 분석하고 디스패처를 제거할 수 있다. 구체적으로, 코드 블록(들)은 블록 내에서 제일 윗줄인 레이블과 제일 아랫줄인 코드 블록 이동 명령문(예를 들어, Goto 문)을 제거한 뒤 배치될 수 있다. 예를 들어, 도 3c의 예시에서 코드 블록(302)은 Label_#1:과 Goto Switch문이 제거되고 "World"만 코드 블록에 남고, 코드 블록(303)은 Label_#2:와 Goto Switch문이 제거되고 "Hello"만 코드 블록에 남고, 코드 블록(304)은 Label_#3:이 제거되고 Exit문만 남은 채로, 그 실행 순서인 코드 블록(303), 코드 블록(302), 코드 블록(304) 순서로 배치될 것이다. 보다 상세하게는, 디스패처(301), 코드 블록(303), 디스패처(301), 코드 블록(302), 디스패처(301), 코드 블록(304) 순서로 배치될 것이다.

[0045] 이후, 배치된 코드 블록에서 디스패처(또는 디스패처 블록)을 제거할 수 있다. 앞서 도 3과 관련하여 설명하였듯이, 디스패처는 평탄화 구조에 적재된 코드 블록 중 실행할 코드 블록을 선택하기 위한 선택 블록으로써 SwitchCase문 등을 의미할 수 있다. 디스패처 블록까지 모두 제거된 후, 코드 블록 재정렬이 완료된다(단계 S507).

[0046] 코드 블록이 재정렬 된 후, 단계 S508에서 알고리즘의 라운드를 1 증가시킨다. 이는 단계 S503으로 다시 돌아가 "아니오"인 경우로 판단되어, 해당 소스코드가 재건이 완료된 소스코드임을 알리기 위함이다.

[0047] 다음, 재건된 난독화된 소스코드는 단계 S509 내지 S512의 소스코드 최적화 단계를 거치게 된다. 일 실시 예에 따르면, 소스코드 제거 모듈(Source Code Elimination Module)(450)과 소스코드 최적화 모듈(Source Code Optimization Module)(460)은 기존의 컴파일러 최적화기에서 적용 가능한 중간 언어에서 구현되던 최적화 기법을 소스코드 레벨에도 적용할 수 있도록 구현될 수 있다.

[0048] 먼저, 삭제된 코드 개수를 의미하는 변수 DeletedCode와 최적화된 코드 개수를 의미하는 변수 OptimizedCode가 초기 값 0으로 설정된다(S509). 그리고 단계 S510에서 소스코드 제거 모듈(450)을 통해, 재건된 소스코드에서 삭제 가능한 패턴을 식별하고 삭제할 수 있다. 일 실시 예에 따르면, 삭제 가능한 패턴은 삭제하여도 코드 블록을 컴파일할 때에 그 기능성에 변동이 없는 소스코드를 의미할 수 있다. 예를 들어, 코드 블록 내에 'A = 3; A = 8;'와 같은 두 줄의 소스코드가 있는 경우, 변수 A는 3으로 저장되었다가 다시 8로 저장될 것이므로, 'A = 3;'을 삭제한 뒤 소스코드를 실행하여도 변수 A가 8로 저장됨에는 변화가 없다. 따라서 이 경우 소스코드 'A = 3;'는 삭제 가능한 패턴일 것이다. 반대로, 코드 블록 내에 'A = 3; B = A + 6; A = 8;'와 같은 세 줄의 소스코드가 있는 경우, 변수 A는 3으로 할당된 뒤에 B의 값 계산에 사용된다. 이 경우, 'A = 3;'을 삭제하고 소스코드를 실행할 시, B의 값이 'A = 3;'가 존재하고 있던 컴파일 결과인 9와 달라질 수 있으므로 코드 블록의 기능성에 변경이 발생하게 된다. 따라서 이 때의 'A = 3;'는 삭제 가능한 코드가 아닐 것이다.

[0049] 다음으로, 단계 S511에서 소스코드 최적화 모듈(460)을 통해, 재건된 소스코드에서 최적화 가능한 패턴을 식별하고 최적화할 수 있다. 이러한 소스코드 제거와 최적화가 수행됨에 따라, 단계 S512에서 DeletedCode는 삭제된 패턴의 수만큼 증가하고 OptimizedCode는 최적화된 패턴의 수만큼 증가하며, 소스코드 최적화가 한번 이루어졌으므로 라운드도 1 증가한다.

[0050] 일 실시 예에 따르면, 소스코드 최적화 단계는 더 이상 제거 및 최적화할 소스코드가 없을 때까지 재건된 코드 블록의 소스코드를 제거 및 최적화하는 과정을 반복할 수 있다. 단계 S513에서 DeletedCode와 OptimizedCode를 합한 값이 0인지 판별할 수 있는데, 이는 더 이상 제거되거나 최적화된 소스코드가 있는지 여부를 확인하기 위함이다. 따라서, 단계 S513에서 "아니오"인 경우, 아직 제거 또는 최적화 할 소스코드가 남아 있거나 더 이상 제거 및 최적화 할 소스코드가 없는지 확인되지 않은 상태이므로 단계 S509로 돌아가 소스코드 최적화 과정을 반복할 것이다. 반대로, 단계 S513에서 "예"인 경우, 더 이상 제거 및 최적화 할 소스코드가 없을 때까지 소스코드가 제거 및 최적화된 것이므로 다음 단계로 넘어가게 된다.

[0051] 소스코드 최적화가 완료된 경우, 단계 S514에서 최적화된 소스코드 생성 모듈(470)을 통해, 최적화된 소스코드를 작성할 수 있다. 사용자는 이러한 소스코드 재건 및 최적화 과정을 거친 소스코드를 분석하고 컴파일 후 실행할 수 있을 것이다.

[0052] 본 발명의 일 실시 예에 따른 소스코드 재건 및 최적화 방법은 컴파일러 최적화기에서 적용할 수 없는 동적 분석의 형태를 적용하고 있기 때문에 더욱 정확한 형태의 제어흐름 재건이 가능하다. 도 6은 일 실시 예에 따른 제어흐름 재건을 나타낸 개념도이다. 도 6을 통해, 제어흐름 평탄화 기술이 적용된 소스코드가 동적 분석을 통해 정확한 제어흐름 재건이 가능한 것을 확인할 수 있다.

[0053] 일 실시 예에 따르면, 프로그램의 실행 순서는 ABAABC 순서일 수 있으나, 제어흐름 평탄화 기술이 적용된 소스

코드의 블록도(610)에서는 컴파일 하지 않는 이상 그 실행 순서를 알 수 없다. 따라서 도 5의 단계 S504 내지 S508의 단계를 통해 재건 기술이 적용된 소스코드의 블록도(620)와 같이 실행 순서대로 코드 블록을 정렬할 수 있다.

[0054] 구체적으로, 레이블의 위치를 파악하여 레이블의 아래에 코드 블록 인덱스를 출력하는 API를 삽입하고 이를 실행할 수 있다. 이 경우, 실제로 실행되는 코드 블록 인덱스만 실행 순서대로 출력될 것이며 출력된 인덱스의 순서대로 코드 블록을 정렬할 수 있다. 그리고 코드 블록 내에서 레이블과 Goto문을 제거하고, 디스패처 블록을 제거하여 코드 블록을 재정렬할 수 있다. 평탄화 구조의 소스코드는 이와 같은 제어흐름 재건 과정을 거친 후 실제 프로그램 실행 순서인 ABAABC 순서대로 코드 블록이 배열될 것이다(620).

[0055] 소스코드 레벨에서 제어흐름의 재건을 선행한 후 소스코드 레벨 최적화 기술을 적용하는 것은, 제어흐름 재건이 선행되고 컴파일러 최적화 옵션이 전혀 적용되지 않은 경우와 비교하였을 때에 더욱 높은 최적화율과 가독성을 제공할 수 있다. 또한, 제어흐름 재건이 선행되지 않고 컴파일 과정에서 컴파일러 최적화 옵션이 적용되는 경우와 비교하였을 때에도 소스코드 레벨에서 제어흐름 재건이 선행되는 경우가 더 높은 가독성을 제공한다. 다시 말해, 제어흐름 난독화 기술이 적용된 소스코드를 타 컴파일러(예를 들어, Visual C++ Compiler, Intel C++ Compiler, GNU Compiler, Collection(GCC), Clang 등)에서 컴파일 할 때 컴파일러 최적화 옵션을 최대로 적용하는 것보다 소스코드 레벨에서 제어흐름 재건과 최적화가 선행된 경우가 명확히 개선된 결과를 보여줄 수 있다.

[0056] 한편, 위와 같은 본 발명의 효과는 일 실시 예에 따라 제어흐름 재건을 수행함으로써 추가 생성되는 최적화 가능 패턴이 존재함을 의미한다. 따라서, 제어흐름 재건이 수행되지 않고 최적화가 수행되는 소스코드보다 제어흐름 재건이 수행되고 최적화가 수행되는 소스코드에서 최적화가 더 많이 이루어지는 것을 알 수 있다.

[0057] 본 발명의 일 실시 예에 따른 소스코드 재건 및 최적화 방법은 또한 소스코드간 변환이기 때문에 컴파일러의 최적화 기술을 적용하였을 때보다 생성 결과물의 가독성이 높을 수 있다. 즉, 컴파일러 최적화 기술은 소스코드가 아닌 중간언어(Intermediate Representation, IP) 레벨에서 적용되기 때문에 생성 결과물 자체가 중간언어 내지는 바이너리에 해당하여 일반적인 분석가에게는 생소할 수 있다. 반면에 본 발명의 일 실시 예에 따른 소스코드 재건 및 최적화 방법은 생성 결과물이 최적화된 소스코드이기 때문에 일반적인 분석가에게도 다소 친숙하다는 점에서 기존 최적화 기술보다 기여도를 가질 수 있다.

[0058] 도 7은 일 실시 예에 따른 소스코드 재건 및 최적화 장치의 블록도이다. 도 1 및 5 등과 같은 순서도에서 각 단계들은 본 발명의 소스코드 재건 및 최적화 장치(700)에 의해 수행될 수 있다. 일 실시 예에 따른 소스코드 재건 및 최적화 장치(700)는 추출부(710), 재건부(720) 및 최적화부(730)를 포함할 수 있다.

[0059] 추출부(710)는 난독화된 소스코드에서 제어흐름 평탄화 기술이 적용된 소스코드 구조를 확인할 수 있다. 보다 상세하게, 추출부(710)는 난독화된 소스코드의 내부를 판독하고 분석하고, 소스코드 내부에서 제어흐름 평탄화 기술이 적용된 평탄화 구조의 위치를 식별하여 추출할 수 있다. 일 실시 예에 따른 추출부(710)는 난독화된 소스코드 파싱 모듈(410) 및 정규 표현식 모듈(420)을 포함할 수 있다.

[0060] 재건부(720)는 확인된 소스코드 구조의 내부에 적재된 코드 블록을 실행 순서에 따라 재건할 수 있다. 보다 상세하게, 재건부(720)는 동적 분석을 적용하여 평탄화 구조 내부의 레이블의 위치를 파악하고, 레이블의 아래에 코드 블록 인덱스를 출력하기 위한 인터페이스를 삽입할 수 있다. 또한, 재건부(720)는 소스코드를 실행하여 출력된 코드 블록 인덱스에 대응하는 코드 블록을 실행 순서대로 배치할 수 있다. 이 때 코드 블록 내에서 레이블과 코드 블록 이동 명령문이 제거되고 디스패처가 제거되어 코드 블록이 실행 순서대로 배열될 수 있다. 일 실시 예에 따른 재건부(720)는 레이블 추출 모듈(430) 및 제어흐름 재건 모듈(440)을 포함할 수 있다.

[0061] 최적화부(730)는 재건된 코드 블록의 제어흐름에 따라 소스코드 최적화를 수행할 수 있다. 보다 상세하게, 최적화부(730)는 재건된 소스코드에서 삭제 가능하거나 최적화 가능한 패턴을 식별하고 삭제 및 최적화할 수 있다. 이는 더 이상 제거 및 최적화할 소스코드가 없을 때까지 반복될 수 있다. 일 실시 예에 따른 최적화부(730)는 소스코드 제거 모듈(450) 및 소스코드 최적화 모듈(460)을 포함할 수 있다.

[0062] 도 7에서는 편의에 따라 추출부(710), 재건부(720) 및 최적화부(730)만을 도시하였으나, 이는 예시일 뿐, 일 실시 예에 따른 소스코드 재건 및 최적화 장치(700)는 다른 구성요소를 더 포함할 수 있다.

[0063] 전술한 실시 예들에 따른 장치는 프로세서, 프로그램 데이터를 저장하고 실행하는 메모리, 디스크 드라이브와 같은 영구 저장부(permanent storage), 외부 장치와 통신하는 통신 포트, 터치 패널, 키(key), 버튼 등과 같은 사용자 인터페이스 장치 등을 포함할 수 있다. 소프트웨어 모듈 또는 알고리즘으로 구현되는 방법들은 상기 프

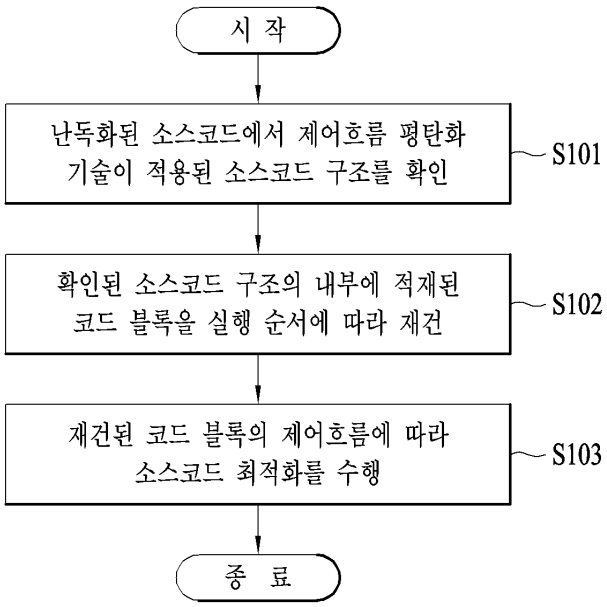
로세서상에서 실행 가능한 컴퓨터가 읽을 수 있는 코드들 또는 프로그램 명령들로서 컴퓨터가 읽을 수 있는 기록 매체 상에 저장될 수 있다. 여기서 컴퓨터가 읽을 수 있는 기록 매체로 마그네틱 저장 매체(예컨대, ROM(read-only memory), RAM(random-Access memory), 플로피 디스크, 하드 디스크 등) 및 광학적 판독 매체(예컨대, 시디롬(CD-ROM), 디브이디(DVD: Digital Versatile Disc)) 등이 있다. 컴퓨터가 읽을 수 있는 기록 매체는 네트워크로 연결된 컴퓨터 시스템들에 분산되어, 분산 방식으로 컴퓨터가 판독 가능한 코드가 저장되고 실행될 수 있다. 매체는 컴퓨터에 의해 판독가능하며, 메모리에 저장되고, 프로세서에서 실행될 수 있다.

[0064] 본 실시 예는 기능적인 블록 구성들 및 다양한 처리 단계들로 나타내어질 수 있다. 이러한 기능 블록들은 특정 기능들을 실행하는 다양한 개수의 하드웨어 또는/및 소프트웨어 구성들로 구현될 수 있다. 예를 들어, 실시 예는 하나 이상의 마이크로프로세서들의 제어 또는 다른 제어 장치들에 의해서 다양한 기능들을 실행할 수 있는, 메모리, 프로세싱, 로직(logic), 룩 업 테이블(look-up table) 등과 같은 직접 회로 구성들을 채용할 수 있다. 구성 요소들이 소프트웨어 프로그래밍 또는 소프트웨어 요소들로 실행될 수 있는 것과 유사하게, 본 실시 예는 데이터 구조, 프로세스들, 루틴들 또는 다른 프로그래밍 구성들의 조합으로 구현되는 다양한 알고리즘을 포함하여, C, C++, 자바(Java), 어셈블러(assembler) 등과 같은 프로그래밍 또는 스크립팅 언어로 구현될 수 있다. 기능적인 측면들은 하나 이상의 프로세서들에서 실행되는 알고리즘으로 구현될 수 있다. 또한, 본 실시 예는 전자적인 환경 설정, 신호 처리, 및/또는 데이터 처리 등을 위하여 종래 기술을 채용할 수 있다. "매커니즘", "요소", "수단", "구성"과 같은 용어는 넓게 사용될 수 있으며, 기계적이고 물리적인 구성들로서 한정되는 것은 아니다. 상기 용어는 프로세서 등과 연계하여 소프트웨어의 일련의 처리들(routines)의 의미를 포함할 수 있다.

[0065] 전술한 실시 예들은 일 예시일 뿐 후술하는 청구항들의 범위 내에서 다른 실시 예들이 구현될 수 있다.

도면

도면1



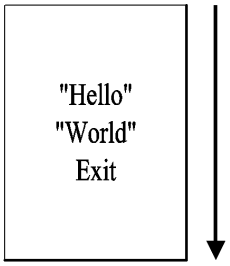
도면2a

```
#include <stdio.h>

int main()
{
    printf("Hello");
    printf("World!\n");
    return 0;
}
```

도면2b

```
; Attributes: bp-based frame

; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near
push    rbp
mov     rbp, rsp
sub     rsp, 20h
call    __main
lea     rcx, aHello     ; "Hello"
call    printf
lea     rcx, aWorld     ; "World!"
call    puts
mov     eax, 0
add     rsp, 20h
pop     rbp
retn
main endp
```

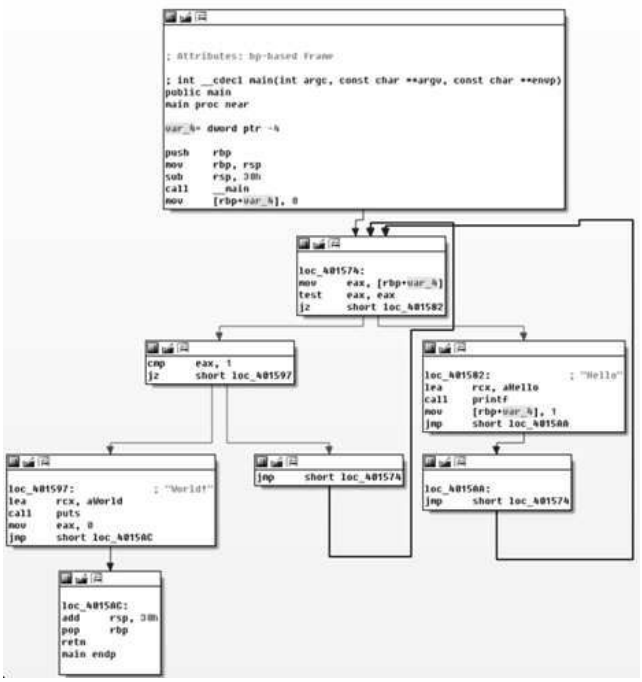
도면2c



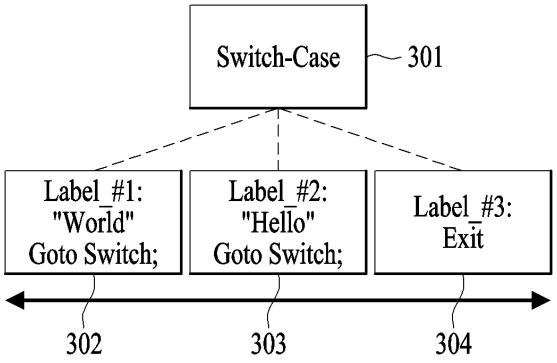
도면3a

```
#include <stdio.h>

int main()
{
    int SwVar = 0;

    while(1)
    {
        switch(SwVar)
        {
        case 0:
            printf("Hello");
            SwVar = 1;
            break;
        case 1:
            printf("World!\n");
            return 0;
        }
    }
}
```

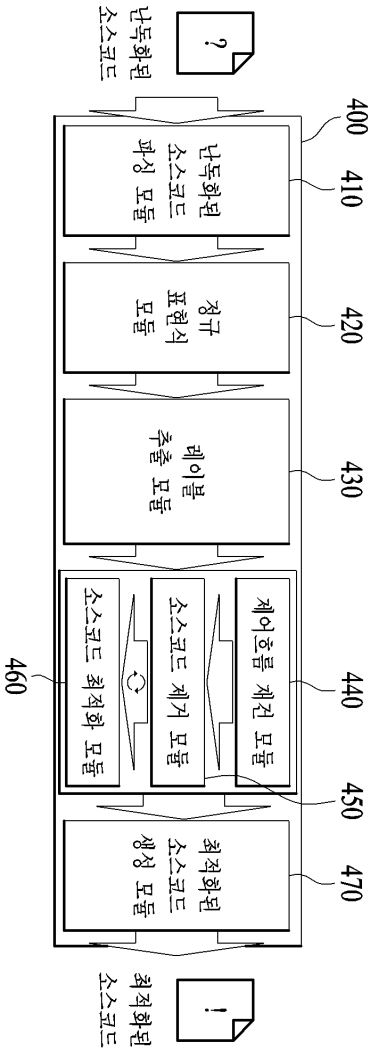
도면3b


; Attributes: bp-based frame
; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near
var_4= dword ptr -4
push rbp
mov rbp, rsp
sub rsp, 30h
call __main
mov [rbp+var_4], 0
loc_401574:
mov eax, [rbp+var_4]
test eax, eax
jz short loc_401582
cmp eax, 1
jz short loc_401597
loc_401582:
; "Hello"
lea rcx, aHello
call printf
mov [rbp+var_4], 1
jmp short loc_4015A0
loc_401597:
; "World!"
lea rcx, aWorld
call puts
mov eax, 0
jmp short loc_4015AC
jmp short loc_401574
loc_4015A0:
jmp short loc_401574
loc_4015AC:
add rsp, 30h
pop rbp
retn
main endp

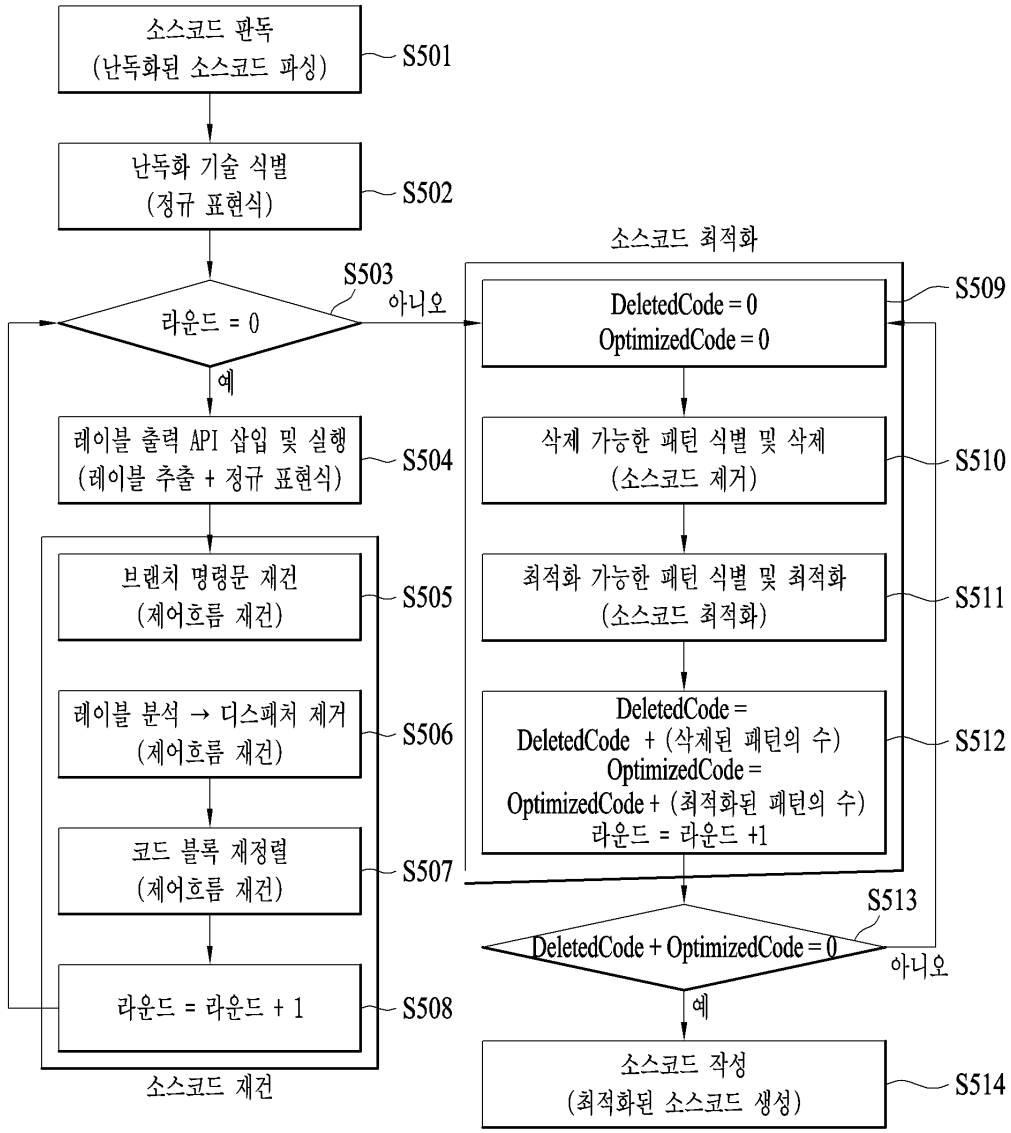
도면3c


Switch-Case
301
Label_#1:
"World"
Goto Switch;
Label_#2:
"Hello"
Goto Switch;
Label_#3:
Exit
302
303
304

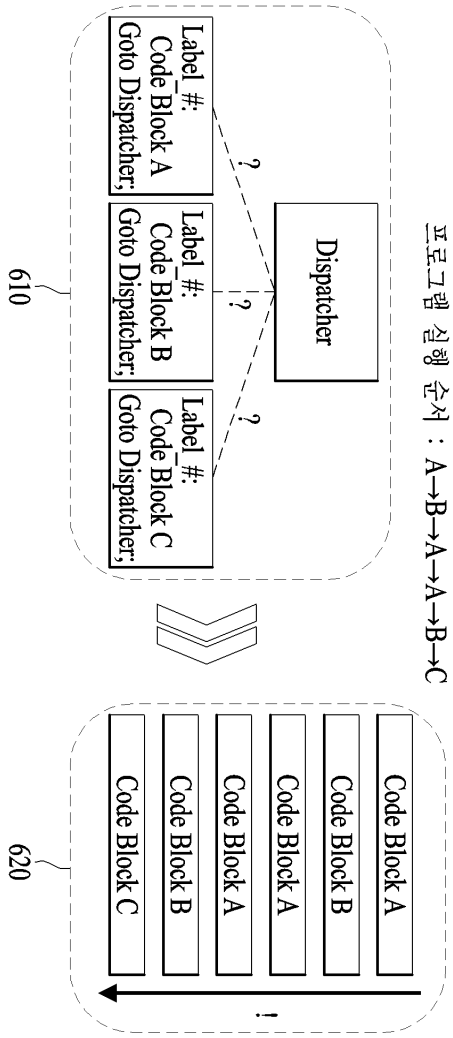
도면4


난독화된 소스코드
?
400
난독화된 소스코드 파싱 모듈
410
정규 표현식 모듈
420
레이블 추출 모듈
430
제어흐름 재건 모듈
440
소스코드 제거 모듈
450
460
소스코드 최적화 모듈
최적화된 소스코드 생성 모듈
470
!
최적화된 소스코드

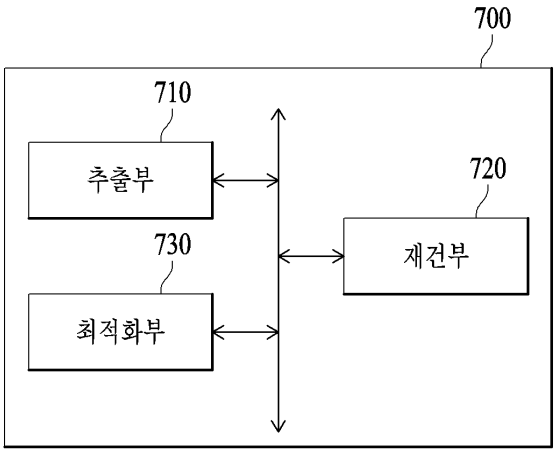
도면5


소스코드 판독
(난독화된 소스코드 파싱)
S501
난독화 기술 식별
(정규 표현식)
S502
S503
라운드 = 0
아니오
예
레이블 출력 API 삽입 및 실행
(레이블 추출 + 정규 표현식)
S504
브랜치 명령문 재건
(제어흐름 재건)
S505
레이블 분석 → 디스패처 제거
(제어흐름 재건)
S506
코드 블록 재정렬
(제어흐름 재건)
S507
라운드 = 라운드 + 1
S508
소스코드 재건
소스코드 최적화
DeletedCode = 0
OptimizedCode = 0
S509
삭제 가능한 패턴 식별 및 삭제
(소스코드 제거)
S510
최적화 가능한 패턴 식별 및 최적화
(소스코드 최적화)
S511
DeletedCode =
DeletedCode + (삭제된 패턴의 수)
OptimizedCode =
OptimizedCode + (최적화된 패턴의 수)
라운드 = 라운드 +1
S512
S513
DeletedCode + OptimizedCode = 0
아니오
예
소스코드 작성
(최적화된 소스코드 생성)
S514

도면6


610
Label_#:
Code Block A
Goto Dispatcher;
Label_#:
Code Block B
Goto Dispatcher;
Label_#:
Code Block C
Goto Dispatcher;
Dispatcher
프로그램 실행 순서 : A→B→A→A→B→C
620
Code Block A
Code Block B
Code Block A
Code Block A
Code Block B
Code Block C

도면7


700
710
추출부
720
재건부
730
최적화부