(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2019-0134336
(43) 공개일자 2019년12월04일

(51) 국제특허분류(Int. Cl.)
G06F 11/07 (2006.01)
(52) CPC특허분류
G06F 11/073 (2013.01)
(21) 출원번호 10-2018-0059843
(22) 출원일자 2018년05월25일
심사청구일자 2018년05월25일

(71) 출원인
고려대학교 산학협력단
서울특별시 성북구 안암로 145, 고려대학교 (안암동5가)
(72) 발명자
이준희
서울특별시 동대문구 안암로18가길 14, A6
홍성준
서울특별시 종로구 난계로29가길 19, 1103호
오학주
서울특별시 성북구 안암로 145, 고려대학교안암캠퍼스 과학도서관 616C
(74) 대리인
김홍석

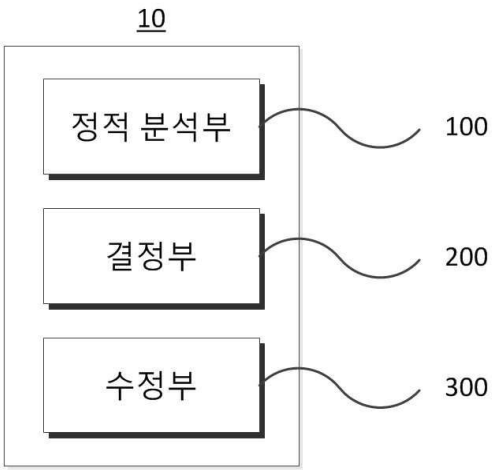
전체 청구항 수 : 총 11 항

(54) 발명의 명칭 **메모리 해제 오류를 자동으로 수정하는 장치 및 방법**

(57) 요 약

메모리 해제 오류 자동 수정 장치가 개시된다. 상기 메모리 해제 오류 자동 수정 장치는 프로그램의 소스 코드에 대하여 정적 분석(static analysis)을 통해 상기 소스 코드에 포함된 객체들 각각에 대하여 상기 객체들이 할당된 위치 정보, 상기 객체들을 가리키는 포인터 정보 및 상기 소스 코드의 각 포인트에 대하여 상기 객체들을 해제할 수 있는 해제문들에 대한 정보인 패치 정보를 포함하는 상태 정보를 생성하는 정적 분석부, 상기 패치 정보 중에서 패치 후보를 선정하고, 상기 객체들을 각각 한 번씩만 해제할 수 있는 상기 패치 후보의 조합을 결정하는 결정부, 및 상기 패치 후보의 조합에 따라 상기 소스 코드를 수정하는 수정부를 포함한다.

대 표 도 - 도2



이 발명을 지원한 국가연구개발사업
과제고유번호 1711065564
부처명 과학기술정보통신부
연구관리전문기관 정보통신기술진흥센터
연구사업명 정보보호핵심원천기술개발
연구과제명 자기학습형 사이버 면역 기술 개발
기 여 율 1/1
주관기관 한국인터넷진흥원
연구기간 2018.01.01 ~ 2018.12.31

명 세 서

청구범위

청구항 1

프로그램의 소스 코드에 대하여 정적 분석(static analysis)을 통해 상기 소스 코드에 포함된 객체들 각각에 대하여 상기 객체들이 할당된 위치 정보, 상기 객체들을 가리키는 포인터 정보 및 상기 소스 코드의 각 포인트에 대하여 상기 객체들을 해제할 수 있는 해제문들에 대한 정보인 패치 정보를 포함하는 상태 정보를 생성하는 정적 분석부;

상기 패치 정보 중에서 패치 후보를 선정하고, 상기 객체들을 각각 한 번씩만 해제할 수 있는 상기 패치 후보의 조합을 결정하는 결정부; 및

상기 패치 후보의 조합에 따라 상기 소스 코드를 수정하는 수정부를 포함하는 메모리 해제 오류 자동 수정 장치.

청구항 2

제1항에 있어서,

상기 정적 분석부는 포인터 분석(point-to analysis)을 통해 상기 포인터 정보를 생성하고,

상기 포인터 정보는 상기 객체들 중 어느 하나의 객체에 대하여 상기 어느 하나의 객체를 가리킬 수 있는 모든 포인터들을 과도 근사화(over-approximate)하여 구한 포인터에 대한 정보인 제1 포인터 정보, 상기 어느 하나의 객체를 가리키는 포인터에 대한 정보인 제2 포인터 정보 및 상기 어느 하나의 객체를 가리키지 않는 포인터에 대한 정보인 제3 포인터 정보를 포함하는 메모리 해제 오류 자동 수정 장치.

청구항 3

제2항에 있어서,

상기 패치 정보는 상기 객체들을 오류 없이 해제할 수 있음이 보장된 해제문들에 대한 정보인 제1 패치 정보 및 상기 객체들을 오류 없이 해제할 수 있음이 보장되지 않은 해제문들에 대한 정보인 제2 패치 정보를 포함하는 메모리 해제 오류 자동 수정 장치.

청구항 4

제3항에 있어서,

상기 정적 분석부는 상기 소스 코드의 각 포인트에 대하여,

상기 각 포인트에서 상기 객체들이 사용될 수 있는 경우 상기 제2 패치 정보에 상기 제1 패치 정보 및 상기 제1 포인터 정보에서 상기 제2 포인터 정보를 제외한 포인터 정보에 대한 해제문들인 제1 해제문들(U)에 대한 정보를 업데이트시키고,

상기 각 포인트에서 상기 객체들이 사용될 수 없는 경우 상기 제2 패치 정보에 상기 제1 해제문들(U)에 대한 정보 및 상기 제2 포인터 정보에 대한 해제문들인 제2 해제문들(G)에 대한 정보와 상기 제2 패치 정보에 동시에 포함되는 해제문들에 대한 정보인 제3 해제문들(D)에 대한 정보를 업데이트시키는 메모리 해제 오류 자동 수정 장치.

청구항 5

제4항에 있어서,

상기 정적 분석부는 상기 소스 코드의 각 포인트에 대하여,

상기 각 포인트에서 상기 객체들이 사용될 수 있는 경우 상기 제1 패치 정보에서 상기 제1 패치 정보에 포함된 해제문들을 제외하고, 상기 제2 해제문들(G)에 대한 정보를 추가하고, 상기 업데이트된 제2 패치 정보에 포함된 해제문들을 제외하여 상기 제1 패치 정보를 업데이트시키고,

상기 각 포인트에서 상기 객체들이 사용될 수 없는 경우 상기 제1 패치 정보에 상기 제2 해제문들(G)에 대한 정보를 추가하고, 상기 업데이트된 제2 패치 정보에 포함된 해제문들을 제외하여 상기 제1 패치 정보를 업데이트시키는 메모리 해제 오류 자동 수정 장치.

청구항 6

제1항에 있어서,

상기 정적 분석부는 상기 소스 코드에 분기문이 포함되어 있는 경우 상기 분기문의 실행 경로에 따라 상기 객체들 각각에 대하여 상기 상태 정보를 생성하는 메모리 해제 오류 자동 수정 장치.

청구항 7

제1항에 있어서,

상기 결정부는 SAT(satisfiability)-솔버를 이용하여 상기 패치 후보의 조합을 결정하는 메모리 해제 오류 자동 수정 장치.

청구항 8

프로그램의 소스 코드에 대하여 정적 분석(static analysis)을 통해 상기 소스 코드에 포함된 객체들 각각에 대하여 상기 객체들이 할당된 위치 정보, 상기 객체들을 가리키는 포인터 정보 및 상기 소스 코드의 각 포인트에 대하여 상기 객체들을 해제할 수 있는 해제문들에 대한 정보인 패치 정보를 포함하는 상태 정보를 생성하는 단계;

상기 패치 정보 중에서 패치 후보를 선정하고, 상기 객체들을 각각 한 번씩만 해제할 수 있는 상기 패치 후보의 조합을 결정하는 단계; 및

상기 패치 후보의 조합에 따라 상기 소스 코드를 수정하는 단계를 포함하는 메모리 해제 오류 자동 수정 방법.

청구항 9

제8항에 있어서,

상기 상태 정보를 생성하는 단계는

포인터 분석(point-to analysis)을 통해 상기 포인터 정보를 생성하는 단계를 더 포함하고,

상기 포인터 정보는 상기 객체들 중 어느 하나의 객체에 대하여 상기 어느 하나의 객체를 가리킬 수 있는 모든 포인터들을 과도 근사화(over-approximate)하여 구한 포인터에 대한 정보인 제1 포인터 정보, 상기 어느 하나의 객체를 가리키는 포인터에 대한 정보인 제2 포인터 정보 및 상기 어느 하나의 객체를 가리키지 않는 포인터에 대한 정보인 제3 포인터 정보를 포함하고,

상기 패치 정보는 상기 객체들을 오류 없이 해제할 수 있음이 보장된 해제문들에 대한 정보인 제1 패치 정보 및 상기 객체들을 오류 없이 해제할 수 있음이 보장되지 않은 해제문들에 대한 정보인 제2 패치 정보를 포함하는 메모리 해제 오류 자동 수정 방법.

청구항 10

제9항에 있어서,

상기 상태 정보를 생성하는 단계는 상기 소스 코드의 각 포인트에 대하여,

상기 각 포인트에서 상기 객체들이 사용될 수 있는 경우 상기 제2 패치 정보에 상기 제1 패치 정보 및 상기 제1 포인터 정보에서 상기 제2 포인터 정보를 제외한 포인터 정보에 대한 해제문들인 제1 해제문들(U)에 대한 정보를 업데이트시키고,

상기 각 포인트에서 상기 객체들이 사용될 수 없는 경우 상기 제2 패치 정보에 상기 제1 해제문들(U)에 대한 정보 및 상기 제2 포인터 정보에 대한 해제문들인 제2 해제문들(G)에 대한 정보와 상기 제2 패치 정보에 동시에 포함되는 해제문들에 대한 정보인 제3 해제문들(D)에 대한 정보를 업데이트시키는 메모리 해제 오류 자동 수정 방법.

청구항 11

제10항에 있어서,

상기 상태 정보를 생성하는 단계는 상기 소스 코드의 각 포인트에 대하여,

상기 각 포인트에서 상기 객체들이 사용될 수 있는 경우 상기 제1 패치 정보에서 상기 제1 패치 정보에 포함된 해제문들을 제외하고, 상기 제2 해제문들(G)에 대한 정보를 추가하고, 상기 업데이트된 제2 패치 정보에 포함된 해제문들을 제외하여 상기 제1 패치 정보를 업데이트시키고,

상기 각 포인트에서 상기 객체들이 사용될 수 없는 경우 상기 제1 패치 정보에 상기 제2 해제문들(G)에 대한 정보를 추가하고, 상기 업데이트된 제2 패치 정보에 포함된 해제문들을 제외하여 상기 제1 패치 정보를 업데이트시키는 메모리 해제 오류 자동 수정 방법.

발명의 설명

기 술 분 야

[0001] 본 발명은 프로그램 자동 수정 장치 및 방법에 관한 것으로서, 보다 구체적으로 메모리 해제 오류를 자동으로 수정하는 장치 및 방법에 관한 것이다.

배 경 기 술

[0002] C/C++와 같은 프로그래밍 언어에서 동적 할당 객체가 올바르게 할당 해제되지 않으면 메모리 해제 오류가 발생한다. 이를 해결하기 위하여는 사용하지 않는 모든 객체들을 수동으로 식별하고 할당 해제하여야 한다. 그러나, 수동으로 메모리 해제 오류를 수정하는 경우 상당한 시간이 소요되고, 수정으로 인한 추가적인 오류가 발생하기 쉽다.

[0003] 프로그램 자동 수정(Automatic-Program-Repair)은 프로그램의 오류를 자동으로 고치는 기술이다. 일반적인 프로그램 자동 수정 기술들은 입출력 예제를 기반으로 동작한다. 보다 구체적으로, 우선 오류를 일으키는 입출력 예제를 통해 오류의 원인을 탐색하고, 올바른 입출력 예제를 만족하도록 프로그램을 수정한다. 그러나, 이러한 기술들은 메모리 해제 오류와 같이 입출력 예제로 표현하기 어려운 오류를 고치기에 적합하지 않다.

[0004] 이에 대하여 비특허문헌 1을 참조하면, 비특허문헌 1은 프로그램에서 메모리 누수임이 확실한 객체에 대하여 해제문(free)을 삽입하는 방식으로 메모리 누수를 고치는 기술을 제안하였다. 그러나, 이러한 기술은 수정된 프로그램이 메모리 누수가 없음을 보장하지 못하며 다른 종류의 메모리 해제 오류들은 고칠 수 없다는 문제점이 있다.

선행기술문헌

비특허문헌

[0005] (비특허문헌 0001) Qing Gao, Yingfei Xiong, Yaqing Mi, Lu Zhang, Weikun Yang, Zhaoping Zhou, Bing Xie, and Hong Mei. 2015. Safe Memory-leak Fixing for C Programs. In Proceedings of the 37th International Conference on Software Engineering (ICSE'15). 459-470.

발명의 내용

해결하려는 과제

[0006] 본 발명의 목적은 소스 코드를 자동으로 검토하고, 메모리 해제 오류가 없도록 수정함으로써 메모리 해제 오류가 없음을 보장하고, 소스 코드를 검토하는 시간을 비약적으로 줄이는데 있다.

과제의 해결 수단

[0007] 본 발명의 일 실시 예에 따른 메모리 해제 오류 자동 수정 장치는 프로그램의 소스 코드에 대하여 정적 분석(static analysis)을 통해 상기 소스 코드에 포함된 객체들 각각에 대하여 상기 객체들이 할당된 위치 정보, 상기 객체들을 가리키는 포인터 정보 및 상기 소스 코드의 각 포인트에 대하여 상기 객체들을 해제할 수 있는 해제문들에 대한 정보인 패치 정보를 포함하는 상태 정보를 생성하는 정적 분석부, 상기 패치 정보 중에서 패치 후보를 선정하고, 상기 객체들을 각각 한 번씩만 해제할 수 있는 상기 패치 후보의 조합을 결정하는 결정부, 및 상기 패치 후보의 조합에 따라 상기 소스 코드를 수정하는 수정부를 포함할 수 있다.

[0008] 본 발명의 일 실시 예에 따른 메모리 해제 오류 자동 수정 방법은 프로그램의 소스 코드에 대하여 정적 분석(static analysis)을 통해 상기 소스 코드에 포함된 객체들 각각에 대하여 상기 객체들이 할당된 위치 정보, 상기 객체들을 가리키는 포인터 정보 및 상기 소스 코드의 각 포인트에 대하여 상기 객체들을 해제할 수 있는 해제문들에 대한 정보인 패치 정보를 포함하는 상태 정보를 생성하는 단계, 상기 패치 정보 중에서 패치 후보를 선정하고, 상기 객체들을 각각 한 번씩만 해제할 수 있는 상기 패치 후보의 조합을 결정하는 단계, 및 상기 패치 후보의 조합에 따라 상기 소스 코드를 수정하는 단계를 포함할 수 있다.

발명의 효과

[0009] 본 발명의 일 실시 예에 따르면, 코드를 자동으로 검토하고, 메모리 해제 오류가 없도록 수정함으로써 코드에 메모리 해제 오류가 없음을 보장할 수 있다.

[0010] 또한, 코드를 검토하는 시간을 비약적으로 줄일 수 있다.

도면의 간단한 설명

[0011] 도 1은 버그 코드 및 버그 코드가 수정된 수정 코드의 예를 도시한 것이다.

도 2는 본 발명의 일 실시 예에 따른 메모리 해제 오류 자동 수정 장치의 블록도이다.

도 3은 본 발명의 일 실시 예에 따른 메모리 해제 오류 자동 수정 방법의 순서도이다.

도 4는 본 발명의 일 실시 예에 따른 정적 분석방법의 순서도이다.

도 5는 정확한 커버 문제의 일 예를 나타낸 것이다.

도 6a는 버그 코드의 일 예를 도시한 것이다.

도 6b는 도 6a의 버그 코드에 대한 정적 분석의 일 예를 도시한 것이다.

도 6c는 도 6a의 버그 코드에 대한 수정된 코드를 도시한 것이다.

도 7a는 벤치마크 프로그램의 일 예를 통한 본 발명의 성능 평가표를 도시한 것이다.

도 7b는 벤치마크 프로그램의 다른 일 예를 통한 본 발명의 성능 평가표를 도시한 것이다.

발명을 실시하기 위한 구체적인 내용

[0012] 본 명세서에 개시되어 있는 본 발명의 개념에 따른 실시예들에 대해서 특정한 구조적 또는 기능적 설명들은 단지 본 발명의 개념에 따른 실시예들을 설명하기 위한 목적으로 예시된 것으로서, 본 발명의 개념에 따른 실시예들은 다양한 형태로 실시될 수 있으며 본 명세서에 설명된 실시예들에 한정되지 않는다.

[0013] 본 발명의 개념에 따른 실시예들은 다양한 변경들을 가할 수 있고 여러 가지 형태들을 가질 수 있으므로 실시예들을 도면에 예시하고 본 명세서에 상세하게 설명하고자 한다. 그러나 이는 본 발명의 개념에 따른 실시예들을 특정한 개시형태들에 대해 한정하려는 것이 아니며, 본 발명의 사상 및 기술 범위에 포함되는 변경, 균등물, 또는 대체물을 포함한다.

[0014] 제1 또는 제2 등의 용어를 다양한 구성요소들을 설명하는데 사용될 수 있지만, 상기 구성요소들은 상기 용어들에 의해 한정되어서는 안 된다. 상기 용어들은 하나의 구성요소를 다른 구성요소로부터 구별하는 목적으로만, 예를 들어 본 발명의 개념에 따른 권리 범위로부터 이탈되지 않은 채, 제1 구성요소는 제2 구성요소로 명명될 수 있고, 유사하게 제2 구성요소는 제1 구성요소로도 명명될 수 있다.

[0015] 어떤 구성요소가 다른 구성요소에 "연결되어" 있다거나 "접속되어" 있다고 언급된 때에는, 그 다른 구성요소에 직접적으로 연결되어 있거나 또는 접속되어 있을 수도 있지만, 중간에 다른 구성요소가 존재할 수도 있다고 이해되어야 할 것이다. 반면에, 어떤 구성요소가 다른 구성요소에 "직접 연결되어" 있다거나 "직접 접속되어" 있다고 언급된 때에는, 중간에 다른 구성요소가 존재하지 않는 것으로 이해되어야 할 것이다. 구성요소들 간의 관계를 설명하는 표현들, 예를 들어 "~사이에"와 "바로~사이에" 또는 "~에 직접 이웃하는" 등도 마찬가지로 해석되어야 한다.

[0016] 본 명세서에서 사용한 용어는 단지 특정한 실시예들을 설명하기 위해 사용된 것으로, 본 발명을 한정하려는 의도가 아니다. 단수의 표현은 문맥상 명백하게 다르게 뜻하지 않는 한, 복수의 표현을 포함한다. 본 명세서에서, "포함하다" 또는 "가지다" 등의 용어는 설시된 특징, 숫자, 단계, 동작, 구성요소, 부분품 또는 이들을 조합한 것이 존재함으로 지정하려는 것이지, 하나 또는 그 이상의 다른 특징들이나 숫자, 단계, 동작, 구성요소, 부분품 또는 이들을 조합한 것들의 존재 또는 부가 가능성을 미리 배제하지 않는 것으로 이해되어야 한다.

[0017] 다르게 정의되지 않는 한, 기술적이거나 과학적인 용어를 포함해서 여기서 사용되는 모든 용어들은 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자에 의해 일반적으로 이해되는 것과 동일한 의미를 가진다. 일반적으로 사용되는 사전에 정의되어 있는 것과 같은 용어들은 관련 기술의 문맥상 가지는 의미와 일치하는 의미를 갖는 것으로 해석되어야 하며, 본 명세서에서 명백하게 정의하지 않는 한, 이상적이거나 과도하게 형식적인 의미로 해석되지 않는다. 이하, 실시예들을 첨부된 도면을 참조하여 상세하게 설명한다.

[0019] 메모리 해제 오류란 C/C++과 같은 프로그래밍 언어에서 개발자가 직접 메모리를 할당한 후에 적절하게 해제하지 않았을 때 생기는 오류이다. 여기에는 ⅰ) 할당된 메모리를 적절하게 해제하지 않았을 경우에 발생하는 메모리 누수(memory-leak), ii) 이미 해제된 메모리를 다시 해제하는 경우에 발생하는 중복 해제(double-free), iii) 할당된 메모리를 너무 일찍 해제하여 해제 후에 다시 사용되는 경우에 발생하는 해제 후 사용(use-after-free) 오류가 있다.

[0021] 도 1은 버그 코드 및 버그 코드가 수정된 수정 코드의 예를 도시한 것이다.

[0022] 도 1을 참조하면, 버그 코드에 해당하는 (a) 코드는 2개의 중복 해제 오류를 포함하고 있다. (a) 코드의 7번째 줄에서 malloc이 실패하고(즉, 버퍼가 재할당되지 않음) 각각 NULL을 반환하면 이미 4번째 줄에서 할당이 해제된 free가 다시 호출된다. 마찬가지로, (a) 코드의 13번째 줄에서 malloc이 실패하면 in은 15번째 줄과 21번째 줄의 free에 의해 두 번에 걸쳐 할당이 해제된다.

[0023] 이에 대하여, (a)의 수정 코드에 해당하는 (b) 코드는 9번째 줄과 16번째 줄에서 버퍼를 NULL화하여 이를 해결하였다. 그러나, free(in)이 중복하여 나타나는 오류 경로가 여전히 남은채 in을 NULL화하는 코드가 삽입되었기 때문에, 해당 프로그램을 올바르게 파악하기 힘들어졌다.

[0024] (b)의 수정 코드에 해당하는 (c) 코드는 개발자가 (b) 코드에 오류가 없음에도 불구하고 메모리 누수 오류가 존재한다고 판단하여 free(out)을 4번째 줄에서 12번째 줄로 이동시켰다. 그러나, (c) 코드에서 in이 재할당에 실패하면, out은 NULL화되고 그로 인해 out 포인터가 가리키는 객체는 더 이상 도달할 수 없게 되어 메모리 누수

오류가 발생하게 된다.

[0025] (c)의 수정 코드에 해당하는 (d) 코드는 (c) 코드의 free(out)을 다시 원래 위치로 이동시켰으며, 중복 해제 오류를 막기 위하여 6번째 줄에서 out을 NULL화 시켰다. 그러나, 해당 코드는 (b) 코드 보다 훨씬 복잡하고 혼란스러워졌다.

[0026] 살펴본 바와 같이 버그 코드의 메모리 해제 오류를 고치는 과정에서 다른 메모리 해제 오류가 발생하거나, 소스 코드가 더욱 복잡해지는 문제가 있다. 이에 대하여, 본 발명의 일 실시 예에 따른 메모리 해제 오류 자동 수정 장치 및 방법을 통하여 버퍼가 재할당되기 바로 직전에 할당 해제함으로써 상술한 문제점을 해결하고자 한다.

[0028] 도 2는 본 발명의 일 실시 예에 따른 메모리 해제 오류 자동 수정 장치의 블록도이다.

[0029] 도 2를 참조하면, 본 발명의 일 실시 예에 따른 메모리 해제 오류 자동 수정 장치(10)는 정적 분석부(100), 결정부(200) 및 수정부(300)를 포함한다.

[0030] 정적 분석부(100)는 입력된 프로그램의 소스 코드에 대하여 정적 분석(static analysis)을 통해 상기 소스 코드에 포함된 객체들 각각에 대하여 상태 정보를 생성한다.

[0031] 상태 정보는 예를 들면 튜플(tuple) 형태로서 다음과 같이 나타내어질 수 있다.

[0032] $$\langle o, may, must, mustNot, patch, patchNot \rangle$$

[0033] 상기 상태 정보에서, o는 상기 객체들이 할당된 위치(allocation-site)에 대한 정보, (may, must, mustNot)은 상기 객체들을 가리키는 포인터 정보, (patch, patchNot)은 상기 소스 코드의 각 포인트에 대하여 상기 객체들을 해제할 수 있는 해제문들에 대한 정보인 패치 정보를 의미한다.

[0034] 일 실시 예에 따른 포인터 정보에 포함된 may는 상기 객체들 중 어느 하나의 객체에 대하여 상기 어느 하나의 객체를 가리킬 수도 있는 포인터들의 집합이다. must는 상기 어느 하나의 객체를 반드시 가리키는 포인터들의 집합이다. mustNot은 상기 어느 하나의 객체를 가리키지 않는 포인터들의 집합이다. 즉, 포인터 정보는 달리 말하면 상기 어느 하나의 객체를 가리킬 수도 있거나, 가리키거나 또는 가리키지 않는 경로들의 집합을 의미한다.

[0035] may에 대하여 보다 상세히 설명하면, 상기 객체들 중 어느 하나의 객체에 대하여 상기 어느 하나의 객체를 가리킬 수 있는 모든 포인터들을 과도 근사화(over-approximate)하여 구한 포인터들의 집합으로서 must를 포함할 수 있으나, mustNot은 제외된다.

[0036] 정적 분석부(100)는 상술한 바와 같은 포인터 정보를 포인터 분석(point-to analysis)을 통하여 생성할 수 있다. 포인터 분석이란 접근경로 기반의 분석으로서 경로 중에 특정 길이를 가지는 경로만 정확히 분석하고, 그 외에 경로가 복잡한 것은 모두 무시하여 근사화하는 분석이다. 포인터 분석은 모든 포인터 변수에 대하여 관계를 찾아내어 포인터 변수가 서로 같은 객체를 가르킬 수 있는지를 분석한다.

[0037] 포인터 분석에는 예를 들면, Andersen analysis 또는 Steensgard analysis 등과 같이 당업자에게 널리 알려진 분석 방법을 채용할 수 있다.

[0038] 상기 패치 정보는 각 객체에 대하여 해제가 가능한 해제문들에 대한 정보를 포함하며, 해제가 가능한 해제문들은 예를 들면 {(c, p)}와 같이 소스 코드의 특정 포인트(c)와 포인터 식(p)으로 나타낼 수 있다. 여기서, {(c, p)}는 소스 코드의 특정 포인트(c) 다음에 삽입되어지는 해제문 free(p)를 의미한다.

[0039] 일 실시 예에 따른 패치 정보에 포함된 patch는 상기 객체들을 메모리 해제 오류 없이, 즉 안전하게 해제할 수 있음이 보장된 해제문들에 대한 정보이다. patchNot은 상기 객체들을 오류 없이 해제할 수 있음이 보장되지 않은 해제문들에 대한 정보이다.

[0040] 정적 분석부(100)는 상기 패치 정보에 포함된 patch와 patchNot을 소스 코드의 각 포인트마다 각각 patch´와 patchNot´으로 업데이트시킨다. 정적 분석부(100)가 패치 정보를 업데이트하는 과정에 대한 상세한 설명은 후술하기로 한다.

[0041] 한편, 정적 분석부(100)는 상기 소스 코드에 분기문이 포함되어 있는 경우 상기 분기문의 실행 경로에 따라 상기 객체들 각각에 대하여 상기 상태 정보를 생성할 수 있다. 예를 들어, 하나의 객체에 대한 상태 정보를 생성하는 경우 분기문의 실행 경로에 따라 true 경로일 때의 객체에 대한 상태 정보와 false 경로일 때의 객체에 대

한 상태 정보를 생성할 수 있다.

[0042] 결정부(200)는 정적 분석부(100)를 통해 생성된 상태 정보에 포함된 패치 정보 중에서 패치 후보를 선정하고, 상기 객체들을 각각 한 번씩만 해제할 수 있는 상기 패치 후보의 조합을 결정한다.

[0043] 패치 후보가 선정되는 패치 정보는 정적 분석부(100)를 통해 소스 코드의 모든 포인트에 대하여 업데이트가 완료된 패치 정보인 (patch´, patchNot´)이다. 여기서, patch´는 각 포인트마다 업데이트된 patch로서 상기 객체들을 안전하게 해제할 수 있는 패치이고, patchNot´는 각 포인트마다 업데이트된 patchNot으로서 상기 객체들을 안전하게 해제할 수 없는 패치이다. 이에 따라, 결정부(200)는 안전한 패치 정보인 patch´에서 안전하지 않은 패치 정보인 patchNot´를 제외하여 패치 후보를 선정할 수 있다.

[0044] 상술한 바와 같이 결정부(200)에 따라 선정된 패치 후보에 따라 소스 코드를 패치하면, 할당된 객체들을 해제 후 사용 오류없이 안전하게 해제시킬 수 있다. 그러나, 상기 패치 후보를 모두 소스 코드에 패치하면 오히려 중복 해제 오류를 발생시킬 수도 있다. 따라서, 할당된 객체들을 중복으로 해제하지 않으면서 각각 한 번씩 모두 해제할 수 있는 해제문들의 조합을 패치 후보로부터 찾아야 한다.

[0045] 일 실시 예에 따른 결정부(200)는 최적의 패치 후보를 찾는 문제를 정확한 커버 문제(Exact Cover Problem)로 변형하고, 이를 해결함으로써 상기 패치 후보로부터 최적의 패치 후보 조합을 찾을 수 있다. 정확한 커버 문제는 NP-완전(NP-complete) 문제로서, 결정부(200)는 다양한 SAT-솔버(SAT-solver)를 이용하여 이를 해결하여 최적의 조합을 찾을 수 있다.

[0046] SAT-솔버는 Boolean 형식으로 주어진 식을 만족하는 해가 있는지 결정하는 문제인 SAT(satisfiability) 문제의 해를 찾는 알고리즘으로서, 예를 들면 Chaff 알고리즘, DPLL 알고리즘, GRASP 알고리즘 및 Satz 알고리즘 등을 포함할 수 있다.

[0047] 수정부(300)는 결정부(200)에서 결정된 패치 후보의 조합에 따라 상기 소스 코드를 수정한다. 수정부(300)는 패치 후보의 조합에 따라 상기 소스 코드의 모든 해제문들을 지우고, 각 포인트에 패치 후보의 해제문들을 삽입하여 소스 코드를 수정하며, 이러한 수정된 소스 코드는 할당된 객체들을 오류없이 한 번씩만 해제하므로 메모리 해제 오류가 없는 것이 보장될 수 있다.

[0049] 도 3은 본 발명의 일 실시 예에 따른 메모리 해제 오류 자동 수정 방법의 순서도이다.

[0050] 도 3을 참조하면, 본 발명의 일 실시 예에 따른 메모리 해제 오류 자동 수정 방법은 소스 코드에 대한 정적 분석을 통해 할당된 모든 객체들에 대한 상태 정보를 생성하고(S100 단계), 생성된 상태 정보로부터 패치 후보를 선정하고, 정확한 커버 문제를 해결하여 상기 모든 객체들을 한 번씩만 해제할 수 있는 패치 후보의 조합을 결정하고(S200 단계), 상기 결정된 패치 후보의 조합에 따라 상기 소스 코드를 수정 하여(S300 단계) 소스 코드에 메모리 해제 오류가 없도록 한다. 이하에서는 S100 단계 및 S200 단계에 대하여 상세히 설명하기로 한다.

[0052] **정적 분석 단계**

[0053] 도 4는 본 발명의 일 실시 예에 따른 정적 분석방법의 순서도이다.

[0054] 도 4를 참조하면, 정적 분석을 수행하는 S100 단계는 포인터 분석을 수행하는 단계(S110 단계) 및 패치 정보를 업데이트하는 단계(S130 단계 내지 S170a, S170b 단계)를 포함한다.

[0055] S110 단계는 포인터 분석을 통하여 소스 코드의 각 포인트(c)에서 아래와 같이 상태 정보를 생성할 수 있다.

[0056] $$\langle o, may', must', mustNot', patch, patchNot \rangle$$

[0057] 여기서, may´는 상기 어느 하나의 객체를 가리킬 수도 있는 포인터들의 집합으로서 상기 상태 정보가 생성되는 소스 코드의 특정 포인트(c) 이전의 포인트에 대한 상태 정보에 포함된 may가 갱신된 것일 수 있다. must´는 상기 어느 하나의 객체를 반드시 가리키는 포인터들의 집합으로서 상기 상태 정보가 생성되는 소스 코드의 특정 포인트(c) 이전의 포인트에 대한 상태 정보에 포함된 must가 갱신된 것일 수 있다. mustNot´은 상기 어느 하나의 객체를 가리키지 않는 포인터들의 집합으로서 상기 상태 정보가 생성되는 소스 코드의 특정 포인트(c) 이전의 포인트에 대한 상태 정보에 포함된 mustNot이 갱신된 것일 수 있다.

[0058] S130 단계 내지 S170a, S170b 단계는 포인터 분석을 통해 생성된 상태 정보를 이용하여 패치 정보를 업데이트시킨다. S130 단계 내지 S170a, S170b 단계에서, 패치 정보는 i) 객체를 해제하지만 해제 후 사용 오류를 일으킬 수 있는 해제문, ii) 객체를 해제하지만 중복 해제 오류를 일으킬 수 있는 해제문, iii) 객체를 해제할 수 있는지 여부가 검증되지 않아 객체의 해제 여부가 불확실한 해제문을 배제하여 아래와 같이 업데이트된다.

[0059] $$\langle o, may', must', mustNot', patch', patchNot' \rangle$$

[0060] 여기서, patch´는 상기 객체들을 메모리 해제 오류 없이, 즉 안전하게 해제할 수 있음이 보장된 해제문들에 대한 정보로서 상기 상태 정보가 생성되는 소스 코드의 특정 포인트(c) 이전의 포인트에 대한 상태 정보에 포함된 patch가 갱신된 것일 수 있다. patchNot´는 상기 객체들을 오류 없이 해제할 수 있음이 보장되지 않은 해제문들에 대한 정보로서 상기 상태 정보가 생성되는 소스 코드의 특정 포인트(c) 이전의 포인트에 대한 상태 정보에 포함된 patchNot이 갱신된 것일 수 있다.

[0061] 보다 상세하게는, patch´와 patchNot´는 아래와 같이 특정 객체(o)에 대하여 상기 특정 객체(o)가 소스 코드의 각 포인트(c)에서 사용될 수 있는지 여부에 따라 아래와 같이 생성될 수 있다.

[0062] $$patch' = \begin{cases} G \setminus patchNot' & o \text{ can be used at } c \\ (patch \cup G) \setminus patchNot' & o \text{ is not used at } c \end{cases}$$

$$patchNot' = \begin{cases} patchNot \cup U \cup patch & o \text{ can be used at } c \\ patchNot \cup U \cup D & o \text{ is not used at } c \end{cases}$$

[0063] 여기서, G는 각 포인트(c)에서 객체를 해제할 수 있는 해제문으로서, 아래와 같다.

[0064] $$G = \{(c,p) \mid p \in must'\}$$

[0065] 즉, 각 포인트(c)에서 객체를 반드시 가리키는 포인터들의 집합인 must´에 속하는 포인터들(p)에 대한 해제문들을 의미한다.

[0066] D는 각 포인트(c)에서 객체를 중복하여 해제하는 해제문으로서, 아래와 같다.

[0067] $$D = patch \cap G$$

[0068] 즉, patch와 G에 동시에 포함되는 해제문들을 의미한다.

[0069] U는 객체의 해제여부가 불확실하여 안정성을 보장할 수 없는 해제문으로서, 아래와 같다.

[0070] $$U = \{(c,p) \mid p \in may' \setminus must'\}$$

[0071] 즉, 각 포인트(c)에서 객체를 가리킬 수도 있는 포인터들의 집합인 may´에서 객체를 반드시 가리키는 포인터들의 집합인 must´을 제외한 포인터들(p)에 대한 해제문들을 의미한다.

[0072] S130 단계 내지 S170a, S170b 단계는 상술한 해제문들(G, D, U)과 기존의 패치 정보(patch, patchNot)를 이용하여 patch´와 patchNot´을 생성한다.

[0073] 이를 위하여 S130 단계에서는 각 포인트(c)에서 특정 객체(o)가 사용될 수 있는지 여부를 판단한다.

[0074] S130 단계의 판단 결과에 따라 만일 객체(o)가 포인트(c)에서 사용될 수 있다면, S150a 단계는 기존의 패치 정보인 patchNot에 patch와 U에 대한 정보를 추가하여 patchNot´으로 업데이트한다.

[0075] 다음으로, S170a 단계는 기존의 패치 정보인 patch에 G에 대한 정보를 추가하고, S150a 단계에서 업데이트된 patchNot´을 제외하여 patch´으로 업데이트한다. 이때, 기존의 patch에 포함된 해제문들은 포인트(c)에서 사용 후 해제 오류를 일으킬 수 있으므로 제외된다.

[0076] 한편, S130 단계의 판단 결과에 따라 만일 객체(o)가 포인트(c)에서 사용될 수 없다면, S150b 단계는 기존의 패치 정보인 patchNot에 U에 대한 정보 및 D에 대한 정보를 추가하여 patchNot´으로 업데이트한다.

[0077] 다음으로, S170b 단계는 기존의 패치 정보인 patch에 G에 대한 정보를 추가하고, (patch ∪ G)에서 S150b 단계

에서 업데이트된 patchNot′을 제외하여 patch′으로 업데이트한다.

[0079] **패치 후보 결정 단계**

[0080] 도 5는 정확한 커버 문제의 일 예를 나타낸 것이다.

[0081] 도 5를 참조하면, S200 단계에서는 도 5와 같은 매트릭스로 나타내어질 수 있는 정확한 커버 문제를 해결하여 상기 모든 객체들을 한 번씩만 해제할 수 있는 패치 후보의 조합을 결정할 수 있다.

[0082] 도 5의 매트릭스에서, 각 행은 선정된 패치 후보의 원소를 나타내고, 각 열은 객체들에 대한 상태 정보를 의미한다. 이때, 매트릭스에 포함된 어떤 원소의 값이 1이라면 원소의 값이 1일 때의 행에 해당하는 패치 후보 {(c, p)}가 해당 열의 상태 정보를 나타내는 객체를 안전하게 해제할 수 있는 것을 의미하고, 원소의 값이 0이라면 원소의 값이 0일 때의 행에 해당하는 패치 후보 {(c, p)}가 해당 열의 상태 정보를 나타내는 객체를 안전하게 해제할 수 없다는 것을 의미한다.

[0083] 예를 들어, 도 5의 매트릭스에서 (1, 2) 성분에 해당하는 원소의 값이 1이므로, 1행의 해제문 {(3, p)}는 2열의 상태 정보(T_2)를 나타내는 객체를 안전하게 해제할 수 있다.

[0084] 한편, 도 5의 매트릭스에서 3열을 참조하면, (2, 3) 성분과 (3, 3) 성분에 해당하는 원소의 값이 1이므로, 3열의 상태 정보(T_3)를 나타내는 객체는 두 번 해제되어 중복 해제 오류가 발생한다.

[0085] 따라서, 모든 객체가 한 번씩만 해제되도록 하는 패치 후보는 도 5의 매트릭스의 각 열에 대하여 1이 오직 1개씩만 있도록하는 행의 집합으로 표현될 수 있으므로, 패치 후보는 {(3, p), (8, q)}이다.

[0086] 상술한 바와 같은 정확한 커버 문제를 해결하기 위하여, S200 단계는 다양한 SAT-솔버를 이용할 수 있다.

[0087] 이하에서는 본 발명의 일 실시 예에 따른 메모리 해제 오류 자동 수정 장치(10) 및 방법이 버그 코드의 일 예에 대하여 상기 버그 코드의 각 포인트별로 상태 정보를 생성 및 업데이트하고 최종적으로 패치 후보를 찾아내는 과정을 설명한다.

[0089] 도 6a는 버그 코드의 일 예를 도시한 것이다.

[0090] 도 6a를 참조하면, 도 6a의 버그 코드는 1->7->9->10 경로에서 중복 해제 오류를 포함한다. 1번째 줄에서, 객체는 할당되고 포인터 p가 상기 객체가 할당된 주소를 가리킨다. 7번째 줄에서, 포인터 q와 p는 같은 객체를 가르킨다. 그러나, 9번째 줄과 10번째 줄에서, 포인터 q와 p가 가리키는 객체는 두 번에 걸쳐 할당해제되므로 중복 해제 오류가 발생한다.

[0092] 도 6b는 도 6a의 버그 코드에 대한 정적 분석의 일 예를 도시한 것이다.

[0093] 도 6b를 참조하면, 도 6b는 도 6a의 버그 코드에 대하여 본 발명의 일 실시 예에 따른 메모리 해제 오류 자동 수정 장치(10) 및 방법에 의해 수행되는 정적 분석에 대한 제어 흐름 그래프(control-flow graph)와 각 포인트에 대하여 생성되는 상태 정보를 함께 나타낸다. 편의상 도 6b에 도시된 정적 분석에 따라 생성되는 상태 정보에서 may는 생략하기로 한다.

[0094] 포인트 1에서, 객체 o1이 할당되고, 포인터 p가 이를 가리킨다. 즉, 해당 객체 o1을 반드시 가리키는 포인터는 {p}이고, 객체 o1을 오류없이 안전하게 해제시킬 수 있는 해제문은 포인트 1 바로 다음에 올 수 있는 해제문 free(p)이다.

[0095] 따라서, 포인트 1에서 생성되는 상태 정보(401)를 참조하면, 할당된 객체 o1에 대한 patch는 {(1, p)}이고, 포인트 1에서는 객체 o1을 반드시 가리키는 포인터만 존재하므로 mustNot과 patchNot은 없다.

[0096] 다음으로 포인트 2의 분기문이 true인 경우, 포인트 3에서 객체 o2가 새로이 할당되고, 포인터 q가 이를 가리킨다. 따라서, 해당 객체 o2를 반드시 가리키는 포인터는 {q}이고, 객체 o2를 오류없이 안전하게 해제시킬 수 있는 해제문은 포인트 3 바로 다음에 올 수 있는 해제문 free(q)이다. 따라서, 포인트 3에서 생성되는 상태 정보(403a)을 참조하면, 할당된 객체 o2에 대한 patch는 {(3, q)}이다.

[0097] 이때, 객체 o2의 할당에 따라 객체 o1에 대한 상태 정보(403b)가 변한다. 포인터 q는 새로이 할당된 객체 o2를 가리키므로 포인터 q가 객체 o1을 절대 가리키지 않음을 확인할 수 있다. 또한, 포인트 3에서도 여전히 free(p)를 삽입하여 객체 o1을 안전하게 해제할 수 있다.

[0098] 따라서, 객체 o1에 대한 상태 정보(403b)에 포함되는 mustNot에 {q}가 업데이트되고, patch는 {(1, p), (3, p)}으로 업데이트된다.

[0099] 한편, 포인트 2의 분기문이 false인 경우, 포인트 7에서 복사 명령문(q = p)에 의해 포인터 q와 p가 모두 객체 o1을 가리키게 됨에 따라, 객체 o1은 포인터 p는 물론 포인터 q에 대한 해제문에 의해서도 해제가 가능하다. 따라서, 포인트 7에서 생성되는 상태 정보(407)를 참조하면, 객체 o1에 대한 patch는 {(1, p), (7, p), (7, q)}이다.

[0100] 분기문이 만나는 지점인 포인트 8 직전에, 정적 분석을 통해 각 분기문에 대한 객체 상태(403a, 403b, 407)을 합쳐 아래와 같은 상태 정보를 생성한다.

[0101]
$$\left\{ \begin{array}{l} \langle o_2, \{q\}, \emptyset, \{(3,q)\}, \emptyset \rangle, \\ \langle o_1, \{p\}, \{q\}, \{(1,p),(3,p)\}, \emptyset \rangle, \\ \langle o_1, \{p,q\}, \emptyset, \{(1,p),(7,p),(7,q)\}, \emptyset \rangle \end{array} \right\}$$

[0102] 상기 상태 정보는 순서대로 분기문이 true일 때와 false일 때의 객체에 대한 상태를 나타낸다. 상기 포인트 8 직전에 생성된 상태 정보는 포인트 8에서의 상태 변화에 따라 업데이트된다.

[0103] 포인트 8에서, 포인터 q가 사용되므로(= *q) q로 접근할 수 있는 모든 객체들에 대한 기존의 patch에 포함된 해제문들은 해제 후 사용 오류를 일으킬 수 있다. 따라서, 기존의 patch에 포함된 해제문들을 patch에서 제거하고, patchNot에 추가한다.

[0104] 포인트 8에서 생성되는 상태 정보(408)를 참조하면, 분기문이 true일 때의 객체 o2에 대한 상태 정보인 τ_1은 기존의 patch에 포함된 {(3, q)}가 사용 후 해제 오류를 일으키므로, 이를 patch에서 제거하여 patchNot에 업데이트하고, 포인트 8에서 안전하게 객체 o2를 해제할 수 있는 {(8, q)}를 patch에 업데이트한다.

[0105] 한편, 분기문이 true일 때의 객체 o1에 대한 상태 정보인 τ_2는 포인터 q가 이를 반드시 가리키지 않으므로, 기존의 patch를 제거하지 않고 {(8, p)}가 업데이트된다.

[0107] 도 6c는 도 6a의 버그 코드에 대한 수정된 코드를 도시한 것이다.

[0108] 도 6c를 참조하면, 도 6b의 정적 분석을 통하여 패치 후보들을 선정하고, 이후에 최적의 패치 후보들의 조합을 결정하여 결정된 조합들에 따라 버그 코드가 수정된다.

[0109] 수정된 코드는 패치 후보들의 조합에 따라 포인트 3 다음의 포인트인 포인트 4에 해제문 free(p)를 삽입하고, 포인트 8 다음의 포인트인 포인트 9에 해제문 free(q)를 삽입함으로써 메모리 해제 오류가 제거된 것을 확인할 수 있다.

[0111] 이하에서는 본 발명의 메모리 해제 오류 수정 성능을 평가하기 위하여, 다른 메모리 누수 오류를 수정하는 도구인 LeakFix(비특허문헌 1)와 오류 수정 성능을 비교하기로 한다. 성능 평가를 위하여, 벤치마크 프로그램인 Juliet Test Suite와 GNU Coreutils를 사용하였다.

[0112] 한편, LeakFix는 메모리 누수 오류만을 수정하는 프로그램이므로, 상기 벤치마크 프로그램에서 해제문을 모두 지우고 메모리 누수 오류를 삽입하여 성능을 비교하기로 한다.

[0114] 도 7a는 벤치마크 프로그램의 일 예를 통한 본 발명의 성능 평가표를 도시한 것이다.

[0115] 도 7a를 참조하면, 도 7a는 벤치마크 프로그램의 일 예인 Juliet Test Suite의 210가지 테스트 사례에 대한 본 발명 및 LeakFix를 적용한 결과를 나타낸다. 결과를 비교하면, LeakFix는 오직 137가지 테스트 사례에 대하여만

패치에 성공하였으나, 본 발명은 모든 테스트 사례에 대하여 패치에 성공하였음을 확인할 수 있다.

[0117] 도 7b는 벤치마크 프로그램의 다른 일 예를 통한 본 발명의 성능 평가표를 도시한 것이다.

[0118] 도 7b를 참조하면, 도 7b는 벤치마크 프로그램의 다른 일 예인 GPU Coreutils의 24가지 프로그램에 대한 본 발명 및 LeakFix를 적용한 결과를 나타낸다. 여기서, #Al.은 프로그램에서 할당 영역의 수를 의미하고, #Ins.는 복구 프로세스가 성공했을 때 각 도구에 의해 삽입된 해제문들의 수를 의미한다. 만약 모든 타겟 할당 영역이 패치에 의해 적절하게 해제된 것으로 볼 수 있으면, 프로그램이 올바르게 수정되었다고 할 수 있다.

[0119] 결과를 비교하면, LeakFix는 24가지 프로그램 중에서 오직 3가지 프로그램에 대하여만 성공적으로 패치하고, 2가지 프로그램은 부분적으로만 패치에 성공한 반면에, 본 발명은 24가지 프로그램 중에서 12가지 프로그램에 대하여 자동으로 패치에 성공하였음을 확인할 수 있다.

[0120] 상기 살펴본 바와 같이, 본 발명은 정적 분석에 의해 생성된 객체들의 상태 정보에 대하여 정확한 커버 문제를 해결함으로써 메모리 해제 오류를 올바르게 수정할 수 있는 패치 후보를 찾을 수 있으며, 이는 기존의 메모리 해제 오류 도구에 비하여 메모리 해제 오류가 없음이 보장됨과 동시에 오류 수정의 성능 또한 더욱 효과적이다.

[0121] 이상에서 설명된 장치는 하드웨어 구성요소, 소프트웨어 구성요소, 및/또는 하드웨어 구성요소 및 소프트웨어 구성요소의 조합으로 구현될 수 있다. 예를 들어, 실시예들에서 설명된 장치 및 구성요소는, 예를 들어, 프로세서, 콘트롤러, ALU(arithmetic logic unit), 디지털 신호 프로세서(digital signal processor), 마이크로컴퓨터, FPA(field programmable array), PLU(programmable logic unit), 마이크로프로세서, 또는 명령(instruction)을 실행하고 응답할 수 있는 다른 어떠한 장치와 같이, 하나 이상의 범용 컴퓨터 또는 특수 목적 컴퓨터를 이용하여 구현될 수 있다. 처리 장치는 운영 체제(OS) 및 상기 운영 체제상에서 수행되는 하나 이상의 소프트웨어 애플리케이션을 수행할 수 있다. 또한, 처리 장치는 소프트웨어의 실행에 응답하여, 데이터를 접근, 저장, 조작, 처리 및 생성할 수도 있다. 이해의 편의를 위하여, 처리 장치는 하나가 사용되는 것으로 설명된 경우도 있지만, 해당 기술분야에서 통상의 지식을 가진 자는, 처리 장치가 복수 개의 처리 요소(processing element) 및/또는 복수 유형의 처리 요소를 포함할 수 있음을 알 수 있다. 예를 들어, 처리 장치는 복수 개의 프로세서 또는 하나의 프로세서 및 하나의 콘트롤러를 포함할 수 있다. 또한, 병렬 프로세서(parallel processor)와 같은, 다른 처리 구성(processing configuration)도 가능하다.

[0122] 소프트웨어는 컴퓨터 프로그램(computer program), 코드(code), 명령(instruction), 또는 이들 중 하나 이상의 조합을 포함할 수 있으며, 원하는 대로 동작하도록 처리 장치를 구성하거나 독립적으로 또는 결합적으로(collectively) 처리 장치를 명령할 수 있다. 소프트웨어 및/또는 데이터는, 처리 장치에 의하여 해석되거나 처리 장치에 명령 또는 데이터를 제공하기 위하여, 어떤 유형의 기계, 구성요소(component), 물리적 장치, 가상 장치(virtual equipment), 컴퓨터 저장 매체 또는 장치, 또는 전송되는 신호 파(signal wave)에 영구적으로, 또는 일시적으로 구체화(embody)될 수 있다. 소프트웨어는 네트워크로 연결된 컴퓨터 시스템 상에 분산되어서, 분산된 방법으로 저장되거나 실행될 수도 있다. 소프트웨어 및 데이터는 하나 이상의 컴퓨터 판독 가능 기록 매체에 저장될 수 있다.

[0123] 실시예에 따른 방법은 다양한 컴퓨터 수단을 통하여 수행될 수 있는 프로그램 명령 형태로 구현되어 컴퓨터 판독 가능 매체에 기록될 수 있다. 상기 컴퓨터 판독 가능 매체는 프로그램 명령, 데이터 파일, 데이터 구조 등을 단독으로 또는 조합하여 포함할 수 있다. 상기 매체에 기록되는 프로그램 명령은 실시예를 위하여 특별히 설계되고 구성된 것들이거나 컴퓨터 소프트웨어 당업자에게 공지되어 사용 가능한 것일 수도 있다. 컴퓨터 판독 가능 기록 매체의 예에는 하드 디스크, 플로피 디스크 및 자기 테이프와 같은 자기 매체(magnetic media), CD-ROM, DVD와 같은 광기록 매체(optical media), 플롭티컬 디스크(floptical disk)와 같은 자기-광 매체(magneto-optical media), 및 롬(ROM), 램(RAM), 플래시 메모리 등과 같은 프로그램 명령을 저장하고 수행하도록 특별히 구성된 하드웨어 장치가 포함된다. 프로그램 명령의 예에는 컴파일러에 의해 만들어지는 것과 같은 기계어 코드뿐만 아니라 인터프리터 등을 사용해서 컴퓨터에 의해서 실행될 수 있는 고급 언어 코드를 포함한다. 상기된 하드웨어 장치는 실시예의 동작을 수행하기 위해 하나 이상의 소프트웨어 모듈로서 작동하도록 구성될 수 있으며, 그 역도 마찬가지이다.

[0124] 이상과 같이 실시예들이 비록 한정된 도면에 의해 설명되었으나, 해당 기술분야에서 통상의 지식을 가진 자라면 상기의 기재로부터 다양한 수정 및 변형이 가능하다. 예를 들어, 설명된 기술들이 설명된 방법과 다른 순서로 수행되거나, 및/또는 설명된 시스템, 구조, 장치, 회로 등의 구성요소들이 설명된 방법과 다른 형태로 결합 또

는 조합되거나, 다른 구성요소 또는 균등물에 의하여 대치되거나 치환되더라도 적절한 결과가 달성될 수 있다.

[0125] 그러므로, 다른 구현들, 다른 실시예들 및 특허청구범위와 균등한 것들도 후술하는 특허청구범위의 범위에 속한다.

도면

도면1

```
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);

in = malloc(2);
if (in == NULL) {

  goto err;
}

out = malloc(2);
if (out == NULL) {
  free(in);

  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

(a) Original code (double-free)

```
in = malloc(1);
out = malloc(1);
...
free(out);
free(in);

in = malloc(2);
if (in == NULL) {
  out = NULL; // +
  goto err;
}

out = malloc(2);
if (out == NULL) {
  free(in);
  in = NULL; // +
  goto err;
}
...
err:
  free(in);
  free(out);
  return;
```

(b) First patch (2007.9) (correct)

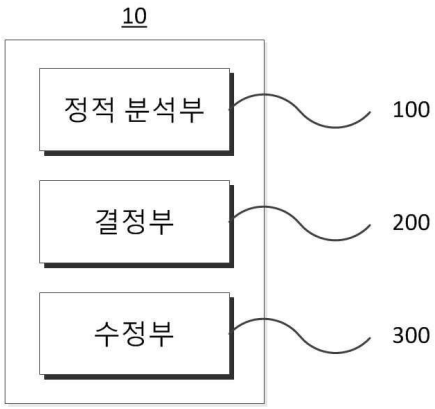
```
in = malloc(1);
out = malloc(1);
...
// -
free(in);

in = malloc(2);
if (in == NULL) {
  out = NULL;
  goto err;
}
free(out); // +
out = malloc(2);
if (out == NULL) {
  free(in);
  in = NULL;
  goto err;
}
...
err:
  free(in);
  free(out);
  return;
```

(c) Second patch (2008.6) (memory-leak)

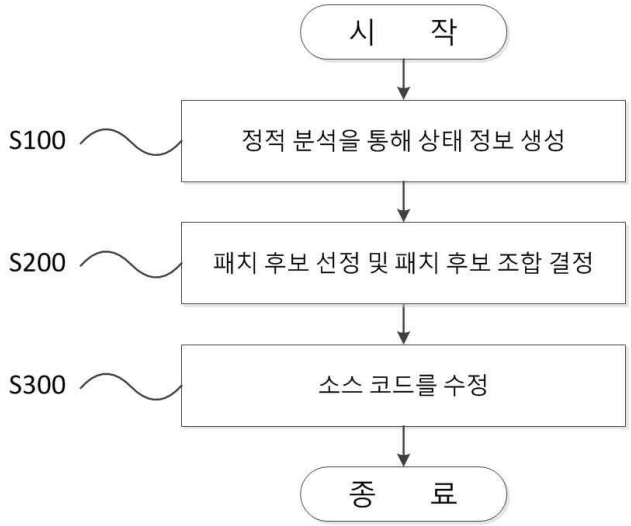
```
in = malloc(1);
out = malloc(1);
...
free(out); // +
free(in);
out = NULL; // +
in = malloc(2);
if (in == NULL) {
  out = NULL;
  goto err;
}
// -
out = malloc(2);
if (out == NULL) {
  free(in);
  in = NULL;
  goto err;
}
...
err:
  free(in);
  free(out);
  return;
```

(d) Third patch (2008.7) (correct)

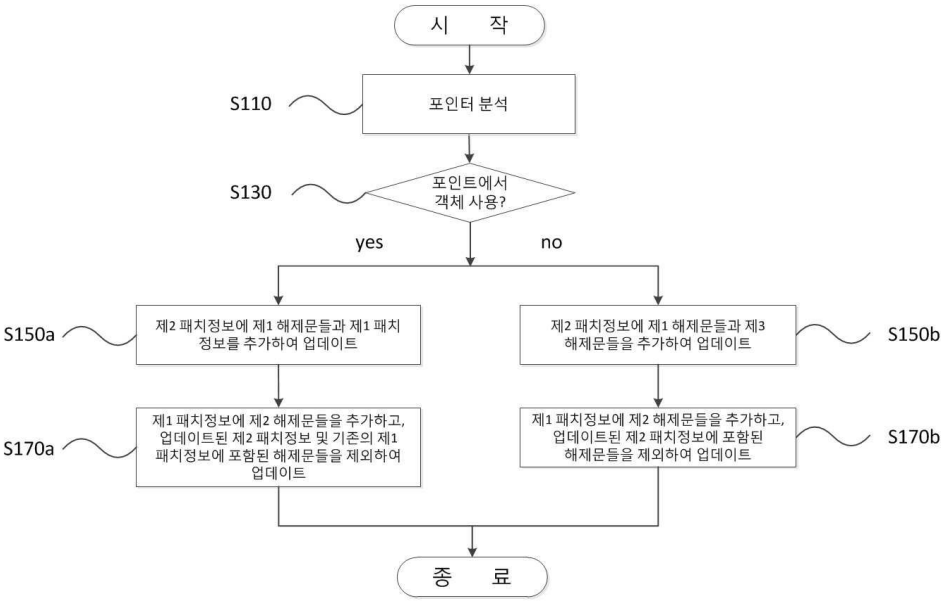
도면2



도면3



도면4



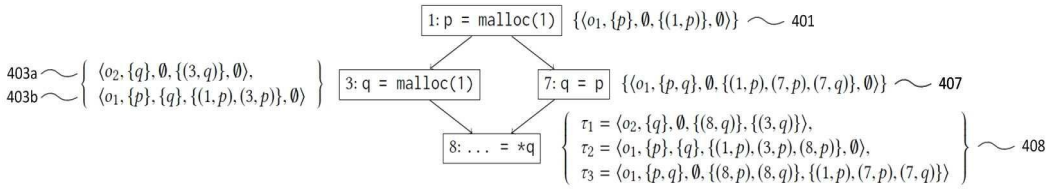
도면5

	τ_1	τ_2	τ_3
$(3, \mathbf{p})$	0	**1**	0
$(8, p)$	0	1	1
$(8, \mathbf{q})$	**1**	0	**1**

도면6a

```
1  p = malloc(1); // o1
2  if (...) {
3    q = malloc(1); // o2
4
5  }
6  else
7     q = p;
8  ... = *q; // use q
9  free(q);
10 free(p);
```

도면6b



도면6c

```
1  p = malloc(1);
2  if (...) {
3    q = malloc(1);
4    free(p); // +
5  }
6  else
7     q = p;
8  ... = *q;
9  free(q);
10 // -
```

도면7a

CWE-ID	Functional Variants	Bugs	Ours	LeakFix
401	int_malloc	38	38	29
	struct_malloc	38	38	29
415	free_int	38	38	20
	free_struct	38	38	19
416	malloc_free_int	20	20	20
	malloc_free_struct	20	20	20
	return_freed_ptr	18	18	0
Total		210	210	137

도면7b

			Ours		LeakFix	
Programs	LoC	#Al.	Fix/Ins.	Time	Fix/Ins.	Time
yes	857	1	1	< 1.0	✗	< 1.0
users	886	1	1	< 1.0	✗	< 1.0
unexpand	1,109	1	1	< 1.0	✗	< 1.0
tee	1,206	1	1	1.3	1	< 1.0
mktemp	1,236	4	✗	1.6	✗	< 1.0
tsort	1,444	3	-	⊥	✗	< 1.0
paste	1,540	3	1	5.9	△/3	< 1.0
date	1,619	1	1	4.3	✗	< 1.0
nl	1,658	4	5	72.4	✗	< 1.0
cut	1,662	1	✗	39.9	✗	< 1.0
pinky	1,740	3	4	27.6	✗	< 1.0
cat	1,891	3	✗	18.7	✗	< 1.0
ln	1,948	2	✗	16.2	✗	< 1.0
printf	2,020	1	1	4.4	✗	< 1.0
stdbuf	2,150	3	3	2.5	✗	< 1.0
wc	2,600	1	1	49.2	△/2	< 1.0
shred	2,836	5	✗	161.4	✗	< 1.0
cp	2,983	8	-	⊥	✗	< 1.0
who	2,990	8	-	⊥	✗	< 1.0
install	3,197	1	-	⊥	✗	< 1.0
tr	3,605	9	-	⊥	✗	< 1.0
expr	3,722	9	-	⊥	✗	< 1.0
stat	3,754	6	3	71.4	✗	< 1.0
dd	5,323	2	-	⊥	✗	< 1.0