(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2021-0045122
(43) 공개일자 2021년04월26일

(51) 국제특허분류(Int. Cl.)
G06F 11/36 (2006.01)
(52) CPC특허분류
G06F 11/3684 (2013.01)
G06F 11/3636 (2013.01)
(21) 출원번호 10-2019-0128481
(22) 출원일자 2019년10월16일
심사청구일자 2019년10월16일

(71) 출원인
연세대학교 산학협력단
서울특별시 서대문구 연세로 50 (신촌동, 연세대학교)
(72) 발명자
권태경
서울특별시 강남구 선릉로 221, 410동 1602호(도곡동, 도곡렉슬아파트)
조민기
서울특별시 서대문구 증가로 26, 302호(연희동)
김서영
서울특별시 서대문구 성산로18길 50, 104호(연희동)
(74) 대리인
특허법인우인

전체 청구항 수 : 총 15 항

(54) 발명의 명칭 **기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치 및 방법**

(57) 요 약

본 실시예들은 기호 실행을 선택적으로 사용하는 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치에 관한 것으로, 프로세서에 의해 실행되는 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치에 있어서, 프로세서는 대상 프로그램 및 입력 데이터의 입력 값을 기반으로 대상 프로그램에서 실행된 명령어의 실행 기록을 저장하는 동적 오염 분석기, 실행 기록 중 일부를 추출하며, 추출된 실행 기록을 분석하여 필드 전이 트리를 생성하는 실행 기록 분석기 및 필드 전이 트리의 다수의 노드들의 깊이를 기반으로 (i) 명령어의 인자 값의 조건을 나타내는 분기 제약 또는 (ii) 명령어의 인자 값을 산출하는 방정식 해석 모듈의 사용 여부를 판단하며, 분기 제약 또는 방정식 해석 모듈을 이용하여 테스트 입력을 생성하는 제약 해석기를 포함하는 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치를 제공한다.

대 표 도 - 도1

(52) CPC특허분류
G06F 11/3688 (2013.01)
G06F 11/3692 (2013.01)

이 발명을 지원한 국가연구개발사업

과제고유번호	2018-0-00513
부처명	과학기술정보통신부
과제관리(전문)기관명	정보통신기획평가원
연구사업명	정보보호핵심원천기술개발사업
연구과제명	기계학습을 활용한 UNIX 기반 커널 취약점 탐지 자동화 연구
기 여 율	1/1
과제수행기관명	연세대학교 산학협력단
연구기간	2019.01.01 ~ 2019.12.31

명 세 서

청구범위

청구항 1

프로세서에 의해 실행되는 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치에 있어서,

상기 프로세서는,

대상 프로그램 및 입력 데이터의 입력 값을 기반으로 상기 대상 프로그램에서 실행된 명령어의 실행 기록을 저장하는 동적 오염 분석기;

상기 실행 기록 중 일부를 추출하며, 상기 추출된 실행 기록을 분석하여 필드 전이 트리를 생성하는 실행 기록 분석기; 및

상기 필드 전이 트리의 다수의 노드들의 깊이를 기반으로 (i) 상기 명령어의 인자 값의 조건을 나타내는 분기 제약 또는 (ii) 상기 명령어의 인자 값을 산출하는 방정식 해석 모듈의 사용 여부를 판단하며, 상기 분기 제약 또는 상기 방정식 해석 모듈을 이용하여 테스트 입력을 생성하는 제약 해석기를 포함하는 것을 특징으로 하는 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치.

청구항 2

제1항에 있어서,

상기 동적 오염 분석기는,

상기 대상 프로그램의 상기 명령어를 실행하고, 상기 명령어의 오염 여부를 판단하는 명령어 추적부; 및

상기 명령어 추적부에서 상기 명령어의 오염을 확인하는 경우 저장부에 저장된 상기 입력 데이터가 다른 저장부로 이동하는 오염 전파를 관리하며, 상기 오염 전파에 의해 데이터 이동 명령어를 제외한 명령어를 실행 라인의 집합으로 형성된 실행 기록에 저장하는 명령어 분석부를 포함하는 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치.

청구항 3

제2항에 있어서,

상기 명령어 추적부는 상기 명령어의 인자 값이 상기 입력 데이터로부터 전달되는 경우 상기 명령어가 오염되었다고 판단하며,

상기 실행 기록의 실행 라인은 (i) 상기 명령어의 주소, (ii) 상기 명령어, (iii) 상기 명령어에 전달된 상기 입력 데이터의 입력 오프셋, (iv) 상기 명령어의 인자 값을 포함하는 것을 특징으로 하는 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치.

청구항 4

제3항에 있어서,

상기 실행 기록 분석기는,

상기 실행 기록 중 일부를 추출하는 실행 기록 최소화부;

상기 추출된 실행 기록에서 상기 입력 오프셋으로부터 필드를 추출하여 상기 필드 별로 분류하는 실행 기록 분류부; 및

상기 필드 별로 분류된 실행 기록을 분석하여 (i) 상기 명령어의 주소, (ii) 상기 명령어, (iii) 상기 인자 값 및 (iv) 상기 명령어가 실행된 결과 값으로 구성된 노드를 포함하는 필드 전이 트리를 생성하는 전이 트리 생성부를 포함하는 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치.

청구항 5

제4항에 있어서,

상기 실행 기록 분류부는 상기 입력 오프셋이 하나 이상 연속적으로 사용되는 경우 하나의 필드인 것으로 판단하여 상기 실행 기록을 분류하는 것을 특징으로 하는 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치.

청구항 6

제4항에 있어서,

상기 전이 트리 생성부는,

상기 필드에 상기 명령어가 실행된 결과 값의 초기 데이터 값을 포함하는 루트 노드에 상기 실행 기록을 기반으로 다수의 노드들이 추가되며,

상기 실행 기록에 저장된 필드의 값과 일치하는 노드 값을 가지는 부모 노드가 상기 트리에 포함되는 경우, 상기 부모 노드에 현재 실행 기록에 저장된 명령어와 상기 저장된 명령어의 실행 결과 값으로 형성된 자식 노드로 추가하여 필드 전이 트리를 생성하는 것을 특징으로 하는 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치.

청구항 7

제5항에 있어서,

상기 제약 해석기는,

상기 필드 전이 트리의 상기 다수의 노드들을 탐색하여 분기 생성에 사용되는 명령어를 포함하는 명령어 노드를 색출하며, 상기 명령어 노드를 기반으로 새로운 경로를 찾기 위한 분기 제약을 생성하는 분기 제약 생성부; 및

새로운 분기문을 실행하기 위해 상기 분기 제약을 해석하며, 상기 분기문의 복잡도에 따라 상기 분기 제약 또는 상기 방정식 해석 모듈의 사용 유무를 결정하는 분기 제약 해석부를 포함하는 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치.

청구항 8

제7항에 있어서,

상기 분기 제약 생성부는 상기 트리의 노드를 탐색하여 상기 명령어 노드를 찾으며,

상기 분기 제약 생성부는 상기 명령어 노드를 발견하는 경우, 루트 노드부터 해당 노드까지 사용된 명령어들을 사용하여 새로운 경로를 찾기 위해 상기 입력 오프셋 및 상기 인자 값을 제약하는 분기 제약을 생성하는 것을 특징으로 하는 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치.

청구항 9

제7항에 있어서,

상기 분기 제약 해석부는,

상기 분기 제약이 생성된 노드의 깊이가 1인 경우, 상기 분기 제약을 이용하여 제약 조건에 만족하는 값을 구하며,

상기 분기 제약이 생성된 노드의 깊이가 2 이상인 경우, 상기 방정식 해석 모듈을 사용하는 것을 특징으로 하는 소프트웨어 테스트 입력 장치.

청구항 10

제7항에 있어서,

상기 제약 해석기는 특정 분기문을 실행에 필요한 상기 입력 오프셋과 상기 인자 값을 기반으로 테스트 입력을 생성하며,

상기 대상 프로그램을 상기 테스트 입력으로 실행하여 생성된 실행 기록을 기반으로 상기 실행 기록 분석기 및 상기 제약 분석기를 반복적으로 실행하여 버그를 확인하는 것을 특징으로 하는 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치.

청구항 11

제1항에 있어서,

상기 대상 프로그램은 랜덤으로 데이터를 입력하여 취약점을 테스트하기 위한 프로그램이고,

상기 입력 데이터는 텍스트 파일, 음성 파일, 영상 파일, 또는 이들이 조합된 데이터인 것을 특징으로 하는 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치.

청구항 12

기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치에 의한 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 방법에 있어서,

대상 프로그램 및 입력 데이터의 입력 값을 기반으로 상기 대상 프로그램에서 실행된 명령어의 실행 기록을 저장하는 단계;

상기 실행 기록 중 일부를 추출하며, 상기 추출된 실행 기록을 분석하여 필드 전이 트리를 생성하는 단계; 및

상기 필드 전이 트리의 다수의 노드들의 깊이를 기반으로 (i) 상기 명령어의 인자 값의 조건을 나타내는 분기 제약 또는 (ii) 상기 명령어의 인자 값을 산출하는 방정식 해석 모듈의 사용 여부를 판단하며, 상기 분기 제약 또는 상기 방정식 해석 모듈을 이용하여 테스트 입력을 생성하는 단계를 포함하는 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 방법.

청구항 13

제12항에 있어서,

상기 명령어의 기록을 남기는 단계는,

상기 대상 프로그램의 상기 명령어를 실행하고, 상기 명령어의 오염 여부를 명령어 추적부에서 판단하는 단계; 및

상기 명령어 추적부에서 상기 명령어의 오염을 확인하는 경우 저장부에 저장된 상기 입력 데이터가 다른 저장부로 이동하는 오염 전파를 관리하며, 상기 오염 전파에 의해 데이터 이동 명령어를 제외한 명령어를 실행 라인의 집합으로 형성된 실행 기록에 저장하는 단계를 포함하고,

상기 실행 기록의 실행 라인은 (i) 상기 명령어의 주소, (ii) 상기 명령어, (iii) 상기 명령어에 전달된 상기 입력 데이터의 입력 오프셋, (iv) 상기 명령어의 인자 값을 포함하는 것을 특징으로 하는 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 방법.

청구항 14

제13항에 있어서,

상기 실행 기록을 분석하여 필드 전이 트리를 생성하는 단계는,

상기 실행 기록 중 일부를 추출하는 단계;

상기 추출된 실행 기록에서 상기 입력 오프셋으로부터 필드를 추출하여 상기 필드 별로 분류하는 단계; 및

상기 필드 별로 분류된 실행 기록을 분석하여 (i) 상기 명령어의 주소, (ii) 상기 명령어, (iii) 상기 인자 값 및 (iv) 상기 명령어가 실행된 결과 값으로 구성된 노드를 포함하는 필드 전이 트리를 생성하는 단계를 포함하는 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 방법.

청구항 15

제12항에 있어서,

상기 제약 조건 값을 생성하는 단계는,

상기 필드 전이 트리의 상기 다수의 노드들을 탐색하여 분기 생성에 사용되는 명령어를 포함하는 명령어 노드를 색출하며, 상기 명령어 노드를 기반으로 새로운 경로를 찾기 위한 분기 제약을 생성하는 단계; 및

새로운 분기문을 실행하기 위해 상기 분기 제약을 해석하며, 상기 분기문의 복잡도에 따라 상기 분기 제약 또는 상기 방정식 해석 모듈의 사용 유무를 결정하는 단계를 포함하는 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 방법.

발명의 설명

기 술 분 야

[0001] 본 발명은 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치 및 방법에 관한 것으로, 특히 기호 실행을 선택적으로 사용하는 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치에 관한 것이다.

배 경 기 술

[0002] 이 부분에 기술된 내용은 단순히 본 실시예에 대한 배경 정보를 제공할 뿐 종래기술을 구성하는 것은 아니다.

[0003] 기존의 기호 실행을 사용한 소프트웨어 테스트 입력 생성 방법들은 사용자 입력이 사용되는 모든 명령어에 대해 기호적 에뮬레이션을 수행하며, 프로그램의 입력 주소에 따른 소프트웨어 테스트 입력 우선순위를 고려하지 않는다. 또한, 기존의 기호 실행을 사용한 소프트웨어 테스트 입력 생성 방법들은 단순한 분기 조건을 해결하기 위해 기호적 에뮬레이션과 SMT 해석기를 사용한다.

[0004] 현재 사용되고 있는 기호 실행을 사용한 테스트 입력 방법은 느린 속도로 인해 규모가 큰 소프트웨어를 테스트하는데 오랜 시간이 걸리는 확장성의 문제를 가지고 있으며, 다수의 문제점을 가지고 있다.

[0005] 기존의 기호 실행을 사용한 테스트 입력 방법은 기호 실행을 위해 사용자 입력에 의해 오염된 모든 명령어들을 기호적으로 에뮬레이션 한다는 것이다. 기존 기호 실행 기술은 데이터 이동을 수행하는 명령어들을 에뮬레이션하기 때문에 속도 저하가 발생한다.

[0006] 또한, 기존의 기호 실행을 사용한 테스트 입력 방법은 중요도가 낮은 분기를 탐색하기 위해 많은 시간을 사용하는 것이다. 프로그램은 입력 데이터를 받아 필요한 정보를 사용한다. 이 때 입력 데이터의 무결성을 검사하거나 인코딩, 디코딩 등이 필요한 경우가 있다. 이와 같은 기능을 수행하는 부분은 단순 데이터 처리이고, 복잡한 로직으로 인해 매우 많은 수의 분기문이 생성되기 때문에 소프트웨어 테스트를 효율적으로 수행하기 위해서는 이 부분을 제외한 부분에 대해 우선적으로 입력을 생성할 필요가 있다.

[0007] 기존의 기호 실행을 사용한 테스트 입력 방법은 단순한 분기를 해결하기 위해 많은 자원을 소모한다는 것이다. 특정 분기문을 실행하기 위한 조건은 복잡할 수도 있고 단순할 수도 있다. 분기문의 조건이 복잡한 경우는 입력의 여러 주소에 의해 분기문의 조건이 결정되거나, 입력의 한 주소의 값이 다른 연산에 의해 변형되어 사용되는 경우가 있고, 분기문의 조건이 단순한 경우는 입력의 한 주소의 값이 다른 연산에 사용되지 않고 분기문의 조건으로 바로 사용되는 경우가 있다. 기존 기호실행 방법은 단순한 분기 조건에 대해서도 기호적 에뮬레이션과 SMT 해석기를 사용하기 때문에 성능 저하가 발생한다.

발명의 내용

해결하려는 과제

[0008] 본 발명의 실시예들은 소프트웨어의 버그를 찾기 위해 기호 실행을 선택적으로 사용하는 기술을 제안한다. 본 발명은 사용자 입력이 사용되는 명령어 중 데이터 이동에 대한 명령어에 대해서는 기호적 에뮬레이션을 수행하지 않기 때문에 더 효율적이며, 넓은 범위의 사용자 입력을 처리하는 명령어들에 대해서는 낮은 우선순위를 둔다.

[0009] 본 발명은 단순한 분기문에 대해서는 기호적 에뮬레이션과 SMT 해석기를 사용하지 않고 직접 분기 조건을 해결하기 때문에 더욱 효율적으로 테스트 입력을 생성한다.

[0010] 본 발명의 명시되지 않은 또 다른 목적들은 하기의 상세한 설명 및 그 효과로부터 용이하게 추론할 수 있는 범

위 내에서 추가적으로 고려될 수 있다.

과제의 해결 수단

[0011] 상기 과제를 해결하기 위해, 본 발명의 일 실시예에 따른 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치는, 프로세서에 의해 실행되는 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치에 있어서, 상기 프로세서는 대상 프로그램 및 입력 데이터의 입력 값을 기반으로 상기 대상 프로그램에서 실행된 명령어의 실행 기록을 저장하는 동적 오염 분석기, 상기 실행 기록 중 일부를 추출하며, 상기 추출된 실행 기록을 분석하여 필드 전이 트리를 생성하는 실행 기록 분석기 및 상기 필드 전이 트리의 다수의 노드들의 깊이를 기반으로 (i) 상기 명령어의 인자 값의 조건을 나타내는 분기 제약 또는 (ii) 상기 명령어의 인자 값을 산출하는 방정식 해석 모듈의 사용 여부를 판단하며, 상기 분기 제약 또는 상기 방정식 해석 모듈을 이용하여 테스트 입력을 생성하는 제약 해석기를 포함한다.

[0012] 여기서, 상기 동적 오염 분석기는 상기 대상 프로그램의 상기 명령어를 실행하고, 상기 명령어의 오염 여부를 판단하는 명령어 추적부 및 상기 명령어 추적부에서 상기 명령어의 오염을 확인하는 경우 저장부에 저장된 상기 입력 데이터가 다른 저장부로 이동하는 오염 전파를 관리하며, 상기 오염 전파에 의해 데이터 이동 명령어를 제외한 명령어를 실행 라인의 집합으로 형성된 실행 기록에 저장하는 명령어 분석부를 포함한다.

[0013] 여기서, 상기 명령어 추적부는 상기 명령어의 인자 값이 상기 입력 데이터로부터 전달되는 경우 상기 명령어가 오염되었다고 판단하며, 상기 실행 기록의 실행 라인은 (i) 상기 명령어의 주소, (ii) 상기 명령어, (iii) 상기 명령어에 전달된 상기 입력 데이터의 입력 오프셋, (iv) 상기 명령어의 인자 값을 포함한다.

[0014] 여기서, 상기 실행 기록 분석기는 상기 실행 기록 중 일부를 추출하는 실행 기록 최소화부, 상기 추출된 실행 기록에서 상기 입력 오프셋으로부터 필드를 추출하여 상기 필드 별로 분류하는 실행 기록 분류부 및 상기 필드 별로 분류된 실행 기록을 분석하여 (i) 상기 명령어의 주소, (ii) 상기 명령어, (iii) 상기 인자 값 및 (iv) 상기 명령어가 실행된 결과 값으로 구성된 노드를 포함하는 필드 전이 트리를 생성하는 전이 트리 생성부를 포함한다.

[0015] 여기서, 상기 실행 기록 분류부는 상기 입력 오프셋이 하나 이상 연속적으로 사용되는 경우 하나의 필드인 것으로 판단하여 상기 실행 기록을 분류한다.

[0016] 여기서, 상기 전이 트리 생성부는 상기 필드에 상기 명령어가 실행된 결과 값의 초기 데이터 값을 포함하는 루트 노드에 상기 실행 기록을 기반으로 다수의 노드들이 추가되며, 상기 실행 기록에 저장된 필드의 값과 일치하는 노드 값을 가지는 부모 노드가 상기 트리에 포함되는 경우, 상기 부모 노드에 현재 실행 기록에 저장된 명령어와 상기 저장된 명령어의 실행 결과 값으로 형성된 자식 노드로 추가하여 필드 전이 트리를 생성한다.

[0017] 여기서, 상기 제약 해석기는 상기 필드 전이 트리의 상기 다수의 노드들을 탐색하여 분기 생성에 사용되는 명령어를 포함하는 명령어 노드를 색출하며, 상기 명령어 노드를 기반으로 새로운 경로를 찾기 위한 분기 제약을 생성하는 분기 제약 생성부 및 새로운 분기문을 실행하기 위해 상기 분기 제약을 해석하며, 상기 분기문의 복잡도에 따라 상기 분기 제약 또는 상기 방정식 해석 모듈의 사용 유무를 결정하는 분기 제약 해석부를 포함한다.

[0018] 여기서, 상기 분기 제약 생성부는 상기 트리의 노드를 탐색하여 상기 명령어 노드를 찾으며, 상기 분기 제약 생성부는 상기 명령어 노드를 발견하는 경우, 루트 노드부터 해당 노드까지 사용된 명령어들을 사용하여 새로운 경로를 찾기 위해 상기 입력 오프셋 및 상기 인자 값을 제약하는 분기 제약을 생성한다.

[0019] 여기서, 상기 분기 제약 해석부는 상기 분기 제약이 생성된 노드의 깊이가 1인 경우, 상기 분기 제약을 이용하여 제약 조건에 만족하는 값을 구하며, 상기 분기 제약이 생성된 노드의 깊이가 2 이상인 경우, 상기 방정식 해석 모듈을 사용한다.

[0020] 여기서, 상기 제약 해석기는 특정 분기문을 실행에 필요한 상기 입력 오프셋과 상기 인자 값을 기반으로 테스트 입력을 생성하며, 상기 대상 프로그램을 상기 테스트 입력으로 실행하여 생성된 실행 기록을 기반으로 상기 실행 기록 분석기 및 상기 제약 분석기를 반복적으로 실행하여 버그를 확인한다.

[0021] 여기서, 상기 대상 프로그램은 랜덤으로 데이터를 입력하여 취약점을 테스트하기 위한 프로그램이고, 상기 입력 데이터는 텍스트 파일, 음성 파일, 영상 파일, 또는 이들이 조합된 데이터이다.

[0022] 본 발명의 또 다른 실시예에 따른 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 방법은, 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치에 의한 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 방법에

있어서, 대상 프로그램 및 입력 데이터의 입력 값을 기반으로 상기 대상 프로그램에서 실행된 명령어의 실행 기록을 저장하는 단계, 상기 실행 기록 중 일부를 추출하며, 상기 추출된 실행 기록을 분석하여 필드 전이 트리를 생성하는 단계 및 상기 필드 전이 트리의 다수의 노드들의 깊이를 기반으로 (i) 상기 명령어의 인자 값의 조건을 나타내는 분기 제약 또는 (ii) 상기 명령어의 인자 값을 산출하는 방정식 해석 모듈의 사용 여부를 판단하며, 상기 분기 제약 또는 상기 방정식 해석 모듈을 이용하여 테스트 입력을 생성하는 단계를 포함한다.

[0023] 여기서, 상기 명령어의 기록을 남기는 단계는 상기 대상 프로그램의 상기 명령어를 실행하고, 상기 명령어의 오염 여부를 명령어 추적부에서 판단하는 단계 및 상기 명령어 추적부에서 상기 명령어의 오염을 확인하는 경우 저장부에 저장된 상기 입력 데이터가 다른 저장부로 이동하는 오염 전파를 관리하며, 상기 오염 전파에 의해 데이터 이동 명령어를 제외한 명령어를 실행 라인의 집합으로 형성된 실행 기록에 저장하는 단계를 포함하고, 상기 실행 기록의 실행 라인은 (i) 상기 명령어의 주소, (ii) 상기 명령어, (iii) 상기 명령어에 전달된 상기 입력 데이터의 입력 오프셋, (iv) 상기 명령어의 인자 값을 포함한다.

[0024] 여기서, 상기 실행 기록을 분석하여 필드 전이 트리를 생성하는 단계는 상기 실행 기록 중 일부를 추출하는 단계, 상기 추출된 실행 기록에서 상기 입력 오프셋으로부터 필드를 추출하여 상기 필드 별로 분류하는 단계 및 상기 필드 별로 분류된 실행 기록을 분석하여 (i) 상기 명령어의 주소, (ii) 상기 명령어, (iii) 상기 인자 값 및 (iv) 상기 명령어가 실행된 결과 값으로 구성된 노드를 포함한다.

[0025] 여기서, 상기 제약 조건 값을 생성하는 단계는 상기 필드 전이 트리의 상기 다수의 노드들을 탐색하여 분기 생성에 사용되는 명령어를 포함하는 명령어 노드를 색출하며, 상기 명령어 노드를 기반으로 새로운 경로를 찾기 위한 분기 제약을 생성하는 단계 및 새로운 분기문을 실행하기 위해 상기 분기 제약을 해석하며, 상기 분기문의 복잡도에 따라 상기 분기 제약 또는 상기 방정식 해석 모듈의 사용 유무를 결정하는 단계를 포함한다.

발명의 효과

[0026] 이상에서 설명한 바와 같이 본 발명의 실시예들에 의하면, 본 발명은 프로그램에서 가장 높은 비율을 가지는 데이터 이동 명령어에 대해 기호적 에뮬레이션을 수행하지 않기 때문에 효율적으로 소프트웨어를 테스트 할 수 있다.

[0027] 본 발명의 실시예들에 의하면, 본 발명은 특정 범위의 입력을 처리하는 명령어에 대해서만 분기 조건을 해결하기 때문에 소프트웨어 버그를 발견하는 실행 분기문을 더 빠르게 찾아갈 수 있다.

[0028] 본 발명의 실시예들에 의하면, 본 발명은 단순한 분기문은 직접 해결하기 때문에 SMT 해석기를 사용하는 다른 기호 실행 방법에 비해 더욱 효율적이다.

[0029] 여기에서 명시적으로 언급되지 않은 효과라 하더라도, 본 발명의 기술적 특징에 의해 기대되는 이하의 명세서에서 기재된 효과 및 그 잠정적인 효과는 본 발명의 명세서에 기재된 것과 같이 취급된다.

도면의 간단한 설명

[0030] 도 1은 본 발명의 일 실시예에 따른 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치를 예시한 블록도이다.

도 2는 본 발명의 일 실시예에 따른 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치를 자세히 예시한 블록도이다.

도 3은 본 발명의 일 실시예에 따른 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치의 동적 오염 분석기의 실행을 나타내는 예시도이다.

도 4는 본 발명의 일 실시예에 따른 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치의 실행 기록 분석기의 필드 전이 트리를 나타내는 예시도이다.

도 5는 본 발명의 일 실시예에 따른 소프트웨어 테스트 입력 생성 방법을 나타내는 흐름도이다.

도 6은 실시예들에서 사용되기에 적합한 컴퓨팅 기기를 포함하는 컴퓨팅 환경을 예시하여 설명하기 위한 블록도이다.

발명을 실시하기 위한 구체적인 내용

[0031] 이하, 본 발명을 설명함에 있어서 관련된 공지기능에 대하여 이 분야의 기술자에게 자명한 사항으로서 본 발명

의 요지를 불필요하게 흐릴 수 있다고 판단되는 경우에는 그 상세한 설명을 생략하고, 본 발명의 일부 실시예들을 예시적인 도면을 통해 상세하게 설명한다. 그러나, 본 발명은 여러 가지 상이한 형태로 구현될 수 있으며, 설명하는 실시예에 한정되는 것이 아니다. 그리고, 본 발명을 명확하게 설명하기 위하여 설명과 관계없는 부분은 생략되며, 도면의 동일한 참조부호는 동일한 부재임을 나타낸다.

[0032] 및/또는 이라는 용어는 복수의 관련된 기재된 항들의 조합 또는 복수의 관련된 기재된 항들 중의 어느 항을 포함한다.

[0033] 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.

[0034] 이하의 설명에서 사용되는 구성요소에 대한 접미사 "모듈" 및 "부"는 명세서 작성의 용이함만이 고려되어 부여되거나 혼용되는 것으로서, 그 자체로 서로 구별되는 의미 또는 역할을 갖는 것은 아니다.

[0035] 제1, 제2 등의 용어는 다양한 구성요소들을 설명하는데 사용될 수 있지만, 구성요소들은 용어들에 의해 한정되어서는 안 된다. 상기 용어들은 하나의 구성요소를 다른 구성요소로부터 구별하는 목적으로만 사용된다.

[0036] 본 발명은 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치 및 방법에 관한 것이다.

[0037] 도 1은 본 발명의 일 실시예에 따른 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치를 예시한 블록도이며, 도 2는 본 발명의 일 실시예에 따른 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치를 자세히 예시한 블록도이다. 도 1에 도시한 바와 같이, 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)는 동적 오염 분석기(100), 실행 기록 분석기(200) 및 제약 해석기(300)를 포함한다. 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)는 도 1에서 예시적으로 도시한 다양한 구성요소들 중에서 일부 구성요소를 생략하거나 다른 구성요소를 추가로 포함할 수 있다.

[0038] 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)는 프로세서 및 프로세서에 의해 실행되는 프로그램을 저장하는 메모리를 포함하며, 프로세서에 의해 동적 오염 분석기(100), 실행 기록 분석기(200) 및 제약 해석기(300)가 수행될 수 있다.

[0039] 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)는 개발 단계에서 찾지 못한 소프트웨어의 버그를 찾기 위한 장치이다. 소프트웨어에 존재하는 버그는 단순히 프로그램의 기능을 사용하지 못하는 것뿐만 아니라 악의적인 공격자에 의해 사용자의 장치가 공격 당할 수 있다.

[0040] 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)는 다른 명령어들의 실행 기록을 통해 데이터의 이동을 확인할 수 있기 때문에 단순 데이터 이동을 수행하는 명령어 (e.g., mov)는 기호적으로 에뮬레이션하지 않아도 된다. 또한, 프로그램은 입력 데이터를 받아 필요한 정보를 사용한다. 이 때 입력 데이터의 무결성을 검사하거나 인코딩, 디코딩 등이 필요한 경우가 있다. 이와 같은 기능을 수행하는 부분은 단순 데이터 처리이고, 복잡한 로직으로 인해 매우 많은 수의 분기문이 생성되기 때문에 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)는 소프트웨어 테스트를 효율적으로 수행하기 위해서 이 부분을 제외한 부분에 대해 우선적으로 입력을 생성할 수 있다.

[0041] 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)는 넓은 범위의 사용자 입력을 처리하는 명령어는 입력데이터의 무결성 검사와 같이 단순 데이터 처리를 위한 명령어일 가능성이 높기 때문에 낮은 우선순위 부여를 통해 소프트웨어의 존재하는 버그를 더욱 효율적으로 발견한다.

[0042] 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)는 데이터 이동을 위한 명령어는 기호 실행을 하지 않고, 특정 범위를 넘어가는 입력을 처리하는 명령어에 대해서는 일부만 기호 실행을 수행하고, 단순한 분기 조건은 직접 해결한다. 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)는 프로그램에서 가장 높은 비율을 가지는 데이터 이동 명령어에 대해 기호적 에뮬레이션을 수행하지 않기 때문에 더욱 효율적이다. 또한, 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)는 특정 범위의 입력을 처리하는 명령어에 대해서만 분기 조건을 해결하기 때문에 소프트웨어의 버그를 발견하는 실행 분기문을 더 빠르게 찾아갈 수 있으며, 단순한 분기문은 직접 해결하기 때문에 SMT (Satisfiability Modulo Theories Solver) 해석기를 사용하는 다른 기호 실행 방법에 비해 더욱 효율적이다.

[0043] 기호 실행은 컴퓨터 프로그램의 입력 값에 대한 실행 경로를 분석하는 방법으로, 입력값에 대한 실행 경로 취득

하며, 역으로 실행 경로를 위한 입력 값을 생성하여 실행 경로를 분석할 수 있다.

[0044] 본 발명의 일 실시예에 따르면, 에뮬레이션은 디지털 정보를 생산한 시점에서 사용된 하드웨어, 매체, 운영 체제, 소프트웨어의 운용을 그대로 흉내 내어(emulate) 한 시스템에서 다른 시스템을 복제하는 방법을 의미한다.

[0045] 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)는 동적 오염 분석기(100)를 사용하여 기록된 명령어 실행 기록을 실행 기록 분석기(200)에서 입력 주소 별로 분석하여 제약 해석기(300)에서 더욱 효율적으로 분기문의 제약 조건을 해결할 수 있다.

[0046] 본 발명의 일 실시예에 따르면, 분기문은 프로그램의 흐름을 조건에 따라서 분기시키며, 프로그램의 순차적 수행 순서에 따르지 않고 다른 명령을 수행하도록 이행 시키는 명령을 의미한다.

[0047] 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)는 사용자의 신뢰를 무너뜨리고 해킹 공격에 악용될 수 있는 소프트웨어의 버그가 발생할 수 있는 프로그램의 실행에 따른 버그를 미리 분석하여, 이를 미연에 해결할 수 있다.

[0048] 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)는 입력 받은 데이터에 의해 실행될 수 있으며, 프로그램이 실행되면서 명령어를 실행시키는 코드가 포함된 코드 라인을 포함하는 실행기록이 저장될 수 있다.

[0049] 본 발명의 일 실시 예에 따른 프로그램 버그와 관련된 입력 데이터는 오염 분석을 이용한 입력 데이터를 나타낼 수 있다.

[0050] 동적 오염 분석기(100)는 대상 프로그램 및 입력 데이터의 입력을 기반으로 대상 프로그램에서 실행된 명령어의 기록을 남길 수 있다.

[0051] 대상 프로그램은 랜덤으로 데이터를 입력하여 취약점을 테스트하기 위한 프로그램일 수 있으며, 입력 데이터는 텍스트 파일, 음성 파일, 영상 파일, 또는 이들이 조합된 데이터일 수 있다.

[0052] 본 발명의 일 실시예에 따르면, 대상 프로그램은 퍼징 대상 프로그램으로, 리눅스용 SO 확장자, 윈도우용 DLL 확장자와 같은 동적 라이브러리 또는 윈도우용 EXE 확장자와 같은 실행 파일로 나타낼 수 있으며, 반드시 이에 한정되는 것은 아니다.

[0053] 본 발명의 일 실시 예에 따른 프로그램은 퍼징(Fuzzing)의 대상이 되는 프로그램일 수 있다. 퍼징은 퍼즈 테스팅(Fuzz testing)이라고도 나타내며, 소프트웨어의 취약점을 테스트하는 방법 중 하나를 나타낸다. 퍼징은 예기치 못한 오류나 크래시(crash)를 일으키도록 소프트웨어에 무작위 데이터, 미리 설정된 데이터 또는 예상치 않은 데이터가 입력되면서 실행되는 테스트이다. 시스템 크래시는 응용 소프트웨어나 운영 체제와 같은 컴퓨터 프로그램이 적절하게 기능하는 것을 멈췄을 때를 나타낸다. 단, 퍼징을 대상으로 하는 프로그램은 본 발명의 일 실시 예를 설명하기 위한 예시일 뿐 이에 한정되는 것이 아니며 보안 문제를 테스트할 수 있는 다양한 프로그램이 사용될 수 있다.

[0054] 본 발명의 일 실시예에 따르면, 입력 데이터는 사용자가 입력하는 데이터일 수 있다. 예를 들어 입력 데이터는 동영상 파일, 텍스트 파일일 수 있으며 반드시 이에 한정되는 것은 아니다. 입력 데이터는 오염으로 판단하는 오염 감염 대상으로, 악의적인 목적으로 버그를 발생시키기 위해 쓰였는지를 확인할 수 있다.

[0055] 본 발명의 일 실시예에 따르면, 대상 프로그램은 입력 데이터를 사용하여 실행될 수 있으며, 대상 프로그램이 실행되면서 명령어를 기반으로 실행 기록이 생성될 수 있다.

[0056] 도 2를 참조하면, 동적 오염 분석기(100)는 대상 프로그램과 입력 데이터를 입력 받아 프로그램에서 실행된 명령어 기록을 남길 수 있다.

[0057] 도 2에 도시한 바와 같이, 동적 오염 분석기(100)는 명령어 추적부(110) 및 명령어 분석부(120)를 포함한다. 동적 오염 분석기(100)는 도 2에서 예시적으로 도시한 다양한 구성요소들 중에서 일부 구성요소를 생략하거나 다른 구성요소를 추가로 포함할 수 있다.

[0058] 본 발명의 일 실시예에 따르면, 동적 오염 분석기(100)는 대상 프로그램과 초기 입력 데이터를 입력 받아 프로그램에서 실행된 명령어 기록을 남길 수 있다.

[0059] 명령어 추적부(110)는 대상 프로그램의 명령어를 실행하며, 상기 명령어의 오염 여부를 판단할 수 있다.

[0060] 명령어 추적부(110)는 명령어의 인자 값이 입력 데이터로부터 전달되는 경우 명령어가 오염되었다고 판단할 수 있다. 예를 들어, 명령어의 인자 값이 입력 데이터로부터 전달되는 경우는 명령어의 인자 값과 입력 데이터가

같은 경우일 수 있다.

[0061] 본 발명의 일 실시예에 따르면, 명령어 추적부(110)는 입력 데이터가 오염되었다고 가정하며, 오염된 인자가 악의적인 목적 혹은 버그를 발생시키는데 쓰였는지를 데이터의 흐름 추적을 통해 탐지하는 방법이다.

[0062] 본 발명의 일 실시예에 따르면, 명령어 추적부(110)는 대상 프로그램의 명령어를 하나씩 실행하고 명령어가 오염되었는지를 검사할 수 있다. 명령어 추적부(110)는 사용자 입력으로부터 명령어의 인자 값이 전달되었다면 명령어가 오염되었다고 판단하며, 실행되는 명령어가 오염되어 있다면 명령어 분석부(120)를 통해 오염 명령어의 분석을 수행한다.

[0063] 명령어 분석부(120)는 명령어 추적부(110)에서 명령어의 오염을 확인하는 경우 저장부에 저장된 입력 데이터가 다른 저장부로 이동하는 오염 전파를 관리하며, 오염 전파에 의해 데이터 이동 명령어를 제외한 명령어를 실행 라인의 집합으로 형성된 실행 기록에 저장할 수 있다.

[0064] 본 발명의 일 실시예에 따르면, 실행 기록은 다수의 실행 라인으로 형성될 수 있으며, 실행 라인의 집합을 나타낸다.

[0065] 본 발명의 일 실시예에 따르면, 저장부는 레지스터 및 메모리를 포함할 수 있으며, 반드시 이에 한정되는 것은 아니다.

[0066] 레지스터는 극히 소량의 데이터나 처리중인 중간 결과를 일시적으로 기억해 두는 고속의 전용 영역을 의미하는 기억 장치이다. 메모리는 정보를 저장하는 기억 장치를 의미한다. 본 발명의 일 실시예에 따르면, 메모리는 실행 명령어를 저장할 수 있다.

[0067] 본 발명의 일 실시예에 따르면, 실행 기록의 실행 라인은 (i) 명령어의 주소, (ii) 명령어, (iii) 명령어에 전달된 입력 데이터의 입력 오프셋, (iv) 명령어의 인자 값을 포함할 수 있으며, 반드시 이에 한정되는 것은 아니다. 본 발명의 일 실시예에 따르면, 명령어는 역어셈블 코드일 수 있다.

[0068] 본 발명의 일 실시예에 따르면, 명령어 분석부(120)는 저장부에 저장된 사용자 입력이 다른 저장부로 이동하는 오염 전파(Taint Propagation)을 관리하고, 현재 명령어의 주소, 역어셈블 코드, 명령어에 전달된 입력 데이터의 입력 오프셋, 명령어의 인자 값을 실행 기록의 실행 라인에 저장할 수 있다. 오염 전파에 의해 데이터의 실행을 확인할 수 있으므로, 실행 기록에는 데이터 이동을 처리하는 mov와 같은 명령어를 제외한 명령어만 저장될 수 있다.

[0069] 본 발명의 일 실시 예에 따르면 명령어의 속성이 복사 명령어일 수 있다. 복사 명령어는 데이터를 복사하기 위해 사용되는 명령어로, 일반적으로 알려진 복사 명령어에는 MOV, LEA와 같은 명령어 종류가 포함될 수 있다. 복사 명령어는 상술한 종류에 한정되지 않으며 데이터를 복사하기 위해 사용되는 모든 명령어가 포함될 수 있다.

[0070] 본 발명의 일 실시 예에 따르면 명령어의 속성이 산술 명령어일 수 있다. 산술 명령어는 산술 연산을 실행하기 위해 사용되는 명령어로, 일반적으로 알려진 산술 명령어에는 ADD, ADC, SUB, RSB, SBC, RSC 또는 MUL과 같은 명령어 종류가 포함될 수 있으며, 산술 명령어는 상술한 종류에 한정되지 않으며 산술 연산을 실행하기 위해 사용되는 모든 명령어가 포함될 수 있다. 또한, 본 발명의 또 다른 일 실시 예에 따르면, 상술한 명령어의 속성이 산술 명령어가 아닌 비교 명령어일 수 있다. 비교 명령어는 두 숫자의 크기를 비교하기 위해 쓰이는 명령어를 나타내며, 상술한 비교 명령어는 순차적으로 진행된 프로그램의 실행 흐름을 조건에 따라 다른 곳으로 분기시키는 분기문을 결정 짓는 명령어를 나타낸다. 따라서, 상술한 비교 명령어의 결과에 따라 분기된 곳에서 하나의 프로그램 실행 경로가 선택되고, 선택되지 않은 프로그램의 실행 경로에서는 프로그램 코드가 실행되지 않는다. 일반적으로 알려진 비교 명령어에는 CMP, TEST 또는 XOR와 같은 명령어 종류가 포함될 수 있으며, 비교 명령어는 상술한 종류에 한정되지 않으며 비교하기 위해 사용되는 모든 명령어가 포함될 수 있다.

[0071] 본 발명의 일 실시예에 따르면, 역어셈블은 기계어 프로그램이 주어졌을 때 그를 분석하여 그의 어셈블리 원시 프로그램을 만들어내는 작업이며, 디버깅, 기존 프로그램의 분석에 이용할 수 있다. 역어셈블은 컴퓨터 코드를 인간이 이해 가능한 어셈블리 언어로 변환할 수 있다.

[0072] 본 발명의 일 실시예에 따르면, 오프셋은 컴퓨터 기억 장치에서 임의 주소에서 간격을 두고 떨어진 주소와의 거리이며, 두 번째 주소를 만들기 위해 기준이 되는 주소에 더해진 값을 의미할 수 있다.

[0073] 동적 오염 분석기(100)의 실행은 도 3을 참조하여 자세히 후술하도록 한다.

[0074] 실행 기록 분석기(200)는 동적 오염 분석기(100)에 의해 프로그램에서 실행되는 실행 기록 중 일부를 추출하며,

추출된 실행 기록을 분석하여 필드 전이 트리를 생성하는 필드 전이 트리를 생성할 수 있다.

[0075] 본 발명의 일 실시예에 따르면, 실행 기록 분석기(200)는 프로그램의 실행 기록을 분석하여 제약 해석기(300)를 효율적으로 사용하기 위한 필드 전이 트리(Field Transition Tree)를 생성할 수 있다. 실행 기록 분석기(200)는 실행 기록 최소화(Execution Trace Reduction), 필드 추론(Field Inference), 필드 전이 트리 생성(Field Transition Tree Construction)을 수행할 수 있다.

[0076] 도 2에 도시한 바와 같이, 실행 기록 분석기(200)는 실행 기록 최소화부(210), 실행 기록 분류부(220) 및 전이 트리 생성부(220)를 포함한다. 실행 기록 분석기(200)는 도 2에서 예시적으로 도시한 다양한 구성요소들 중에서 일부 구성요소를 생략하거나 다른 구성요소를 추가로 포함할 수 있다.

[0077] 실행 기록 최소화부(210)는 명령어에 의해 생성된 실행 기록 중 일부를 추출할 수 있다. 본 발명의 일 실시예에 따르면, 실행 기록의 추출은 랜덤 추출, 상위 몇 개 추출 또는 하위 몇 개를 추출할 수 있으며, 반드시 이에 한정되는 것은 아니며 사용자에 의해 설정될 수 있다.

[0078] 예를 들어, 실행 기록 최소화부(210)는 실행 기록을 짧은 것 위주로 선별하며, 긴 것은 일부만 확인할 수 있다. 예를 들어, 실행 기록이 긴 것은 10개만 확인할 수 있으며, 확인하는 개수는 사용자에 의해 설정될 수 있다.

[0079] 실행 기록 최소화부(210)는 생성된 실행 기록 중 일부를 추출할 수 있으며, 또는 넓은 범위의 입력을 사용하는 명령어의 가중치를 낮게 조정하여 실행 기록을 최소화할 수 있다. 실행 기록을 최소화하는 방법은 상술한 바에 한정되지 않는다.

[0080] 본 발명의 일 실시예에 따르면, 실행 기록 최소화부(210)는 넓은 범위의 입력을 사용하는 명령어들에 의해 생성된 실행 기록을 일부만 남기는 최소화 과정이 이루어질 수 있다.

[0081] 실행 기록 분류부(220)는 추출된 실행 기록에서 명령어의 인자로 사용된 입력 오프셋으로부터 필드를 추출하여 필드 별로 분류할 수 있다.

[0082] 실행 기록 분류부(220)는 입력 오프셋이 하나 이상 연속적으로 사용되는 경우 하나의 필드인 것으로 판단하여 분류할 수 있다. 예를 들어, 연속적으로 사용되는 경우는 입력 오프셋이 같은 것끼리 하나의 필드로 묶을 수 있으며, 또는 입력 오프셋이 일정한 간격에 따라 연속적으로 사용되는 경우를 의미할 수 있으며 반드시 이에 한정되는 것은 아니다.

[0083] 본 발명의 일 실시예에 따르면, 실행 기록 분류부(220)는 실행 기록을 통해 최소화부(210)에서 추출된 최소화된 실행 기록에서 명령어의 인자로 사용된 입력 오프셋으로부터 필드를 추출할 수 있다. 실행 기록 분류부(220)는 입력 오프셋이 하나 이상 연속적으로 사용되는 경우에 하나의 필드인 것으로 판단하고, 같은 필드를 인자로 사용하는 입력 기록들을 분류할 수 있다.

[0084] 전이 트리 생성부(230)는 필드 별로 분류된 실행 기록을 분석하여 (i) 명령어의 주소, (ii) 명령어, (iii) 인자 값 및 (iv) 명령어가 실행된 결과 값으로 구성된 노드를 포함하는 필드 전이 트리를 생성할 수 있다.

[0085] 전이 트리 생성부(230)는 필드에 입력 데이터의 명령어가 실행된 결과 값을 포함하는 루트 노드에 상기 실행 기록을 기반으로 다수의 노드들이 추가되며, 실행 기록에 저장된 필드의 값과 일치하는 노드 값을 가지는 부모 노드가 트리에 포함되는 경우, 부모 노드에 현재 실행 기록에 저장된 명령어와 저장된 명령어의 실행 결과 값을 명령어와 입력 데이터의 데이터 값을 가지는 자식 노드로 추가하여 필드 전이 트리를 생성할 수 있다.

[0086] 본 발명의 일 실시예에 따르면, 노드는 주소, 명령어, 제1 인자 값, 제2 인자 값, 명령어의 실행 결과 값으로 구성될 수 있으며, 반드시 이에 한정되는 것은 아니다.

[0087] 본 발명의 일 실시예에 따르면, 전이 트리 생성부(230)는 필드별로 분류된 분류 기록들을 분석하여 각 필드의 필드 전이 트리를 생성한다. 필드 전이 트리는 명령어와 값으로 구성된 노드들로 구성될 수 있다.

[0088] 본 발명의 일 실시예에 따른 필드 전이 트리는 도 3을 참조하여 자세히 후술하도록 한다.

[0089] 제약 해석기(300)는 실행 기록 분석기(200)의 필드 전이 트리의 다수의 노드들의 깊이를 기반으로 (i) 명령어의 인자 값의 조건을 나타내는 분기 제약 또는 (ii) 명령어의 인자 값을 산출하는 방정식 해석 모듈의 사용 여부를 판단하며, 분기 제약 또는 방정식 해석 모듈을 이용하여 테스트 입력을 생성할 수 있다.

[0090] 본 발명의 일 실시예에 따르면, 방정식 해석 모듈은 방정식의 해를 구하기 위한 모듈이다. 예를 들어 방정식 해석 모듈은 SMT (Satisfiability Modulo Theories Solver) 해석기일 수 있으며, 본 발명의 일 실시예를 설명하

기 위한 예시일 뿐 이에 한정되는 것은 아니며 방정식의 해를 구하는 다양한 도구들이 사용될 수 있다.

[0091] 방정식 해석 모듈은 수식을 만족하는 값이 존재하는지 찾아준다. 수식을 만족하는 값이 존재할 경우 그 값을 구해준다. 방정식 해석 모듈은 일차 방정식 또는 부등식으로 형성된 수식의 값을 산출할 수 있다.

[0092] 제약 해석기(300)는 특정 분기문을 실행에 필요한 입력 오프셋과 인자 값을 기반으로 테스트 입력을 생성하며, 대상 프로그램을 테스트 입력으로 실행하여 생성된 실행 기록을 기반으로 실행 기록 분석기 및 제약 분석기를 반복적으로 실행하여 상기 프로그램의 버그를 확인할 수 있다.

[0093] 본 발명의 일 실시예에 따르면, 제약 해석기(300)는 실행 기록 분석기(200)의 전이 트리 생성부(230)에서 생성된 필드 전이 트리로부터 분기 제약을 생성하고 이에 대한 해석을 수행하며, 이를 위해 방정식 해석 모듈을 사용할 수 있다.

[0094] 도 2에 도시한 바와 같이, 제약 해석기(300)는 분기 제약 생성부(310) 및 분기 제약 해석부(320)를 포함한다. 제약 해석기(300)는 도 2에서 예시적으로 도시한 다양한 구성요소들 중에서 일부 구성요소를 생략하거나 다른 구성요소를 추가로 포함할 수 있다.

[0095] 분기 제약 생성부(310)는 필드 전이 트리의 다수의 노드들을 탐색하여 분기 생성에 사용되는 명령어를 포함하는 명령어 노드를 색출하며, 명령어 노드를 기반으로 새로운 경로를 찾기 위한 분기 제약을 생성할 수 있다.

[0096] 분기 제약 생성부(310)는 트리의 노드를 탐색하여 명령어 노드를 찾을 수 있다. 분기 제약 생성부(310)는 명령어 노드를 발견하는 경우, 루트 노드부터 해당 노드까지 사용된 명령어들을 사용하여 새로운 경로를 찾기 위한 분기 제약을 생성할 수 있다.

[0097] 본 발명의 일 실시예에 따르면, 분기 제약 생성부(310)는 트리의 노드 탐색하며 분기 생성에 사용되는 cmp 등의 명령어가 있는 노드를 찾을 수 있다. 분기 제약 생성부(310)는 분기 명령어 노드를 발견하면 루트 노드부터 해당 노드까지 사용된 명령어들을 사용하여 새로운 경로를 찾기 위한 분기 제약을 생성할 수 있다.

[0098] 분기 제약 해석부(320)는 새로운 분기문을 실행하기 위한 분기 제약을 해석하며, 분기문의 복잡도에 따라 방정식 해석 모듈의 사용 유무를 결정할 수 있다.

[0099] 분기 제약 해석부(320)는 분기 제약이 생성된 노드의 깊이가 1인 경우, 방정식 해석 모듈을 사용하지 않고 분기 제약 조건에 만족하는 값을 구하며, 분기 제약이 생성된 노드의 깊이가 2 이상인 경우, 방정식 해석 모듈을 사용할 수 있다.

[0100] 본 발명의 일 실시예에 따르면, 분기 제약 해석부(320)는 새로운 분기문을 실행하기 위한 제약을 해석하며, 분기문의 복잡도에 따라 방정식 해석 모듈을 사용 유무를 결정할 수 있다.

[0101] 본 발명의 일 실시예에 따르면, 분기 제약 해석부(320)는 분기 제약이 생성된 노드의 깊이가 1일 경우 방정식 해석 모듈을 사용하지 않고 제약 조건에 만족하는 값을 구하며 노드의 깊이가 2 이상일 경우 방정식 해석 모듈을 사용하며, 일 실시예일뿐 반드시 이에 한정되지 않는다.

[0102] 본 발명의 일 실시예에 따르면, 제약 해석기(300)는 특정 분기문을 실행에 필요한 입력 오프셋과 인자 값을 알아내어 테스트 입력을 생성하며, 대상 프로그램을 테스트 입력으로 실행하여 생성된 실행 기록을 기반으로 실행 기록 분석기 및 제약 분석기를 반복적으로 실행하여 프로그램에서 발생하는 버그를 확인할 수 있다.

[0103] 도 3은 본 발명의 일 실시예에 따른 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치의 동적 오염 분석기의 실행을 나타내는 예시도이다.

[0104] 도 3은 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)의 동적 오염 분석기(100)의 실행을 예시적으로 나타낸 도면이다. 도 3a는 동적 오염 분석기(100)의 실행 코드를 나타내며, 도 3b는 도 3a의 실행 코드가 실행되면 형성되는 실행 기록을 나타내는 예시도이다.

[0105] 본 발명의 일 실시예에 따르면, 도 3a의 왼쪽 부분의 코드 중 While의 조건에 해당하는 cmp 명령어가 실행되면 도 3b와 같은 실행 기록이 저장될 수 있다.

[0106] 본 발명의 일 실시예에 따르면, 도 3a의 S는 입력 데이터를 나타낼 수 있다.

[0107] 도 3b를 참조하면, 실행 기록에 포함된 각각의 코드 라인은 명령어 주소(102), 명령어(104), 첫 번째 인자의 입력 오프셋과 인자 값(106) 및 두 번째 인자의 입력 오프셋과 인자 값(108)을 나타낼 수 있다.

[0108] 본 발명의 일 실시예에 따르면, 명령어 주소(102)는 프로그램에서 명령어의 주소를 나타낸 것으로, "0x80c3a07"을 나타낸다.

[0109] 본 발명의 일 실시예에 따르면, 명령어(104)는 "cmp"를 나타내며, 역어셈블된 명령어를 나타낸다. 명령어(104)는 기계어를 어셈블된 명령어로 변환하는 것을 나타낸다. 기계어는 컴퓨터가 읽을 수 있는 2진 숫자로 이루어진 언어를 나타내며, 어셈블리된 명령어는 기계어를 사람이 보기 쉽게 문자를 기호화한 것을 나타낼 수 있다.

[0110] 명령어(104)는 비교 명령어에 해당하는 "cmp"를 나타내며, 비교 명령어의 종류 중 하나인 "cmp"는 본 발명의 일 실시예를 설명하기 위한 예시일 뿐 반드시 이에 한정되는 것은 아니며 다양한 비교 명령어로 형성될 수 있다. 비교 명령어는 순차적으로 진행된 프로그램의 실행 흐름을 조건에 따라 다른 곳으로 분기시키는 분기문을 결정짓는 역어셈블된 명령어를 나타낼 수 있다.

[0111] 본 발명의 일 실시예에 따르면, 첫 번째 인자의 입력 오프셋과 인자 값(106)은 첫 번째 인자의 입력 오프셋(106a)과 첫 번째 인자의 인자 값(106b)으로 형성될 수 있다. 첫 번째 인자의 입력 오프셋(106a)은 "{00,-,-,-}"을 나타낸다. 첫 번째 인자의 인자 값(106b)은 "{61,00,00,00}"을 나타낸다.

[0112] 본 발명의 일 실시예에 따르면, 첫 번째 인자의 인자 값(106b)의 61은 명령어의 연산을 수행하면서 바뀔 수 있는 값이다.

[0113] 본 발명의 일 실시예에 따르면, 두 번째 인자의 입력 오프셋과 인자 값(108)은 두 번째 인자의 입력 오프셋(108a)과 두 번째 인자의 인자 값(108b)으로 형성될 수 있다. 두 번째 인자의 입력 오프셋(108a)은 "{-,-,-,-}"을 나타낸다. 두 번째 인자의 인자 값(108b)은 "{00,00,00,00}"을 나타낸다.

[0114] 본 발명의 일 실시예에 따르면, 첫 번째 인자의 입력 오프셋(106a) 및 두 번째 인자의 입력 오프셋(108a)은 명령어의 인자에 전달된 입력 데이터의 입력 오프셋을 의미할 수 있다. 또한, 첫 번째 인자의 인자 값(106b) 및 두 번째 인자의 인자 값(108b)은 명령어의 인자로 사용되는 인자 값을 의미할 수 있다.

[0115] 본 발명의 일 실시예에 따르면, 인자 값은 명령어의 인자 값으로, 레지스터에 저장된 값을 나타내는 레지스터 값, 메모리 주소를 그대로 쓰는 값 또는 메모리에 저장된 값을 나타내는 메모리 값 및 명령어 자체에서 표시된 상수 값을 의미할 수 있다. 상수 값은 고정된 값이며, 레지스터 값 및 메모리 값은 가변적인 값이다.

[0116] 본 발명의 일 실시예에 따르면, 첫 번째 인자의 입력 오프셋과 인자 값(106) 및 두 번째 인자의 입력 오프셋과 인자 값(108)은 레지스터 값, 메모리 값 및 상수 값 중 하나의 값으로 형성될 수 있으며, 반드시 이에 한정되는 것은 아니다.

[0117] 본 발명의 일 실시예에 따르면, 입력 오프셋은 실행 기록에 포함된 각각의 코드 라인에 포함된 명령어를 실행시키는데 사용된 입력 데이터가 기록된 주소를 나타낸 것일 수 있다.

[0118] 본 발명의 일 실시예에 따르면, 첫 번째 인자의 입력 오프셋과 인자 값(106) 및 두 번째 인자의 입력 오프셋과 인자 값(108)은 각 인자가 4 바이트 크기이기 때문에 4개의 입력 오프셋과 인자 값이 저장되며 저장되는 입력 오프셋과 인자 값의 수는 명령어에서 사용되는 인자의 크기에 따라 달라질 수 있다. 인자의 크기는 본 발명의 일 실시예를 설명하기 위한 예시일 뿐 반드시 이에 한정되는 것은 아니다.

[0119] 도 4는 본 발명의 일 실시예에 따른 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치의 실행 기록 분석기의 필드 전이 트리를 나타내는 예시도이다.

[0120] 도 4는 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)의 실행 기록 분석기(200)의 전이 트리 생성부(230)에서 생성된 필드 전이 트리를 예시적으로 나타낸 도면이다.

[0121] 노드는 트리(Tree) 구조에서 데이터의 상하위 계층을 나타내는 위치의 항목을 의미하며, 각 노드를 연결해 주는 것이 분기이며, 가장 상위 노드를 나타내는 루트(Root)와 하위 노드들로 구성된 서브트리(Subtree)로 구성되며, 데이터를 주고받는 모든 시스템을 통칭할 수 있다.

[0122] 본 발명의 일 실시예에 따르면, 필드 전이 트리는 루트(302)를 나타내는 R 노드(302) "Value = 0 x 100"와 A 노드(304), B 노드(306), C 노드(308), D 노드(309)를 포함하는 서브트리로 형성될 수 있으며, 하위 노드들은 반드시 이에 한정되는 것은 아니다.

[0123] 본 발명의 일 실시예에 따르면, R 노드(302)의 "Value = 0 x 100"는 0 x 100의 주소에서 하나의 명령어가 실행된 결과 값을 나타낸다.

[0124] 본 발명의 일 실시예에 따르면, A 노드(304)는 명령어의 메모리상 주소를 나타내는 "addr1", 명령어를 나타내는 "cmp", 제1 인자 값을 나타내는 "X", 제2 인자 값을 나타내는 "0 x 200" 및 제1 입력 오프셋이며 X의 값을 나타내는 "Value = 0 x 100"을 포함한다. 예를 들어 A 노드(304)의 자식 노드인 B 노드(306)는 제1 입력 오프셋이며 X의 값을 나타내는 "Value = 0 x 100"이 X 값에 대입되어 명령어 "add"에 의해 더해진 값인 "Value = 110"으로 형성될 수 있다.

[0125] 도 4를 참조하면, 깊이(Depth)는 필드 전이 트리에서 부모에서 자식순으로 이동할 때, 1씩 증가하고, 형제 노드 간의 깊이가 일치한다. 루트(302)의 깊이는 0이 된다.

[0126] 도 4를 참조하면, 필드 전이 트리의 B 노드에서는 명령어 "add"를 사용하며, C 노드에서는 명령어 "mul"을 사용한다. 산술 명령어를 나타내는 "add", "mul"은 계산 결과 값이 기 설정된 경계 값을 초과하도록 만드는 산술 명령어이다.

[0127] 본 발명의 일 실시예에 따르면, 필드 전이 트리는 필드의 입력 오프셋의 값을 가지고 아무런 명령어가 존재하지 않는 루트 노드만 가지고 있는 상태로 초기화되며 입력된 실행 기록에 따라 노드들이 추가될 수 있다.

[0128] 본 발명의 일 실시예에 따르면, 전이 트리 생성부(230)는 실행 기록에 저장된 필드의 값과 같은 노드 값을 가지는 노드가 트리에 있다면 해당 노드에 현재 실행 기록에 저장된 명령어와 명령어의 실행 결과값을 노드의 명령어와 값으로 가지는 새로운 노드를 자식 노드로 추가한다. 만약 트리에 같은 노드 값을 가지는 노드가 2개 이상 존재한다면 마지막에 추가된 노드의 자식 노드로 추가한다. 필드 값과 같은 노드 값을 가지는 노드가 존재하지 않는다면 해당 실행 기록은 노드에 추가되지 않을 수 있다.

[0129] 제약 해석기(300)는 상술한 필드 전이 트리를 기반으로 분기 제약을 형성하고, 분기 제약에 대한 해석을 수행할 수 있다.

[0130] 본 발명의 일 실시예에 따르면, D 노드(309)에 대한 분기 제약은 A 노드(304), C 노드(308), D 노드(309)에 의해 생성될 수 있다. D 노드(309)에 대한 분기 제약은 A 노드(304)에 의해 "X < 0 x 200" 및 C 노드(308) 및 D 노드(309)에 의해 "X * Y < 0 x 100" 와 같은 제약이 생길 수 있다.

[0131] 분기 제약 중 하나인 "X < 0 x 200 및 X * Y < 0 x 100"은 본 발명의 일 실시예를 설명하기 위한 예시일 뿐 반드시 이에 한정되는 것은 아니다.

[0132] 본 발명의 일 실시예에 따르면, 다수의 X 값 및 Y 값이 산출될 수 있으며, 새로운 테스트 입력을 생성할 수 있다.

[0133] 이하에서는 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)를 통한 소프트웨어 테스트 입력 생성 과정에 대해 흐름을 따라 설명한다.

[0134] 도 5는 본 발명의 일 실시예에 따른 소프트웨어 테스트 입력 생성 방법을 나타내는 흐름도이다. 소프트웨어 테스트 입력 생성 방법은 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치에 의하여 수행될 수 있으며, 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치가 수행하는 동작에 관한 상세한 설명과 중복되는 설명은 생략하기로 한다.

[0135] 소프트웨어 테스트 입력 생성 방법은 대상 프로그램 및 입력 데이터의 입력 값을 기반으로 대상 프로그램에서 실행된 명령어의 실행 기록을 저장하는 단계(S510), 실행 기록 중 일부를 추출하며, 추출된 실행 기록을 분석하여 필드 전이 트리를 생성하는 단계(S520), 필드 전이 트리의 다수의 노드들의 깊이를 기반으로 (i) 명령어의 인자 값의 조건을 나타내는 분기 제약 또는 (ii) 명령어의 인자 값을 산출하는 방정식 해석 모듈의 사용 여부를 판단하며, 분기 제약 또는 방정식 해석 모듈을 이용하여 테스트 입력을 생성하는 단계(S530)를 포함한다.

[0136] 대상 프로그램 및 입력 데이터의 입력 값을 기반으로 대상 프로그램에서 실행된 명령어의 실행 기록을 저장하는 단계(S510)는 대상 프로그램의 명령어를 실행하고, 명령어의 오염 여부를 명령어 추적부에서 판단하는 단계 및 명령어 추적부에서 명령어의 오염을 확인하는 경우 저장부에 저장된 입력 데이터가 다른 저장부로 이동하는 오염 전파를 관리하며, 오염 전파에 의해 데이터 이동 명령어를 제외한 명령어를 실행 라인의 집합으로 형성된 실행 기록에 저장하는 단계를 포함할 수 있다.

[0137] 실행 기록의 실행 라인은 (i) 명령어의 주소, (ii) 명령어, (iii) 명령어에 전달된 상기 입력 데이터의 입력 오프셋, (iv) 명령어의 인자 값을 포함한다.

[0138] 실행 기록 중 일부를 추출하며, 추출된 실행 기록을 분석하여 필드 전이 트리를 생성하는 단계(S520)는 실행 기

록 중 일부를 추출하는 단계, 추출된 실행 기록에서 입력 오프셋으로부터 필드를 추출하여 필드 별로 분류하는 단계 및 필드 별로 분류된 실행 기록을 분석하여 (i) 명령어의 주소, (ii) 명령어, (iii) 인자 값 및 (iv) 명령어가 실행된 결과 값으로 구성된 노드를 포함하는 필드 전이 트리를 생성하는 단계를 포함한다.

[0139] 실행 기록 중 일부를 추출하며, 필드 전이 트리의 다수의 노드들의 깊이를 기반으로 (i) 명령어의 인자 값의 조건을 나타내는 분기 제약 또는 (ii) 명령어의 인자 값을 산출하는 방정식 해석 모듈의 사용 여부를 판단하며, 분기 제약 또는 방정식 해석 모듈을 이용하여 테스트 입력을 생성하는 단계(S530)는 필드 전이 트리의 다수의 노드들을 탐색하여 분기 생성에 사용되는 명령어를 포함하는 명령어 노드를 색출하며, 명령어 노드를 기반으로 새로운 경로를 찾기 위한 분기 제약을 생성하는 단계 및 새로운 분기문을 실행하기 위해 분기 제약을 해석하며, 분기문의 복잡도에 따라 분기 제약 또는 방정식 해석 모듈의 사용 유무를 결정하는 단계를 포함한다.

[0140] 도 5에서는 각각의 과정을 순차적으로 실행하는 것으로 개재하고 있으나 이는 예시적으로 설명한 것에 불과하고, 이 분야의 기술자라면 본 발명의 실시예의 본질적인 특성에서 벗어나지 않는 범위에서 도 5에 기재된 순서를 변경하여 실행하거나 또는 하나 이상의 과정을 병렬적으로 실행하거나 다른 과정을 추가하는 것으로 다양하게 수정 및 변형하여 적용 가능할 것이다.

[0141] 도 6은 예시적인 실시예들에서 사용되기에 적합한 컴퓨팅 기기를 포함하는 컴퓨팅 환경을 예시하여 설명하기 위한 블록도이다. 도시된 실시예에서, 각 컴포넌트들은 이하에 기술된 것 이외에 상이한 기능 및 능력을 가질 수 있고, 이하에 기술되지 것 이외에도 추가적인 컴포넌트를 포함할 수 있다.

[0142] 도시된 컴퓨팅 환경은 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)를 포함한다. 일 실시예에서, 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)는 타 단말기와 신호를 송수신하는 모든 형태의 컴퓨팅 기기일 수 있다.

[0143] 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)는 적어도 하나의 프로세서(610), 컴퓨터 판독 가능한 저장매체(620) 및 통신 버스(660)를 포함한다. 프로세서(610)는 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)로 하여금 앞서 언급된 예시적인 실시예에 따라 동작하도록 할 수 있다. 예컨대, 프로세서(610)는 컴퓨터 판독 가능한 저장 매체(620)에 저장된 하나 이상의 프로그램들을 실행할 수 있다. 상기 하나 이상의 프로그램들은 하나 이상의 컴퓨터 실행 가능 명령어를 포함할 수 있으며, 상기 컴퓨터 실행 가능 명령어는 프로세서(610)에 의해 실행되는 경우 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)로 하여금 예시적인 실시예에 따른 동작들을 수행하도록 구성될 수 있다.

[0144] 컴퓨터 판독 가능한 저장 매체(620)는 컴퓨터 실행 가능 명령어 내지 프로그램 코드, 프로그램 데이터 및/또는 다른 적합한 형태의 정보를 저장하도록 구성된다. 컴퓨터 판독 가능한 저장 매체(620)에 저장된 프로그램(630)은 프로세서(610)에 의해 실행 가능한 명령어의 집합을 포함한다. 일 실시예에서, 컴퓨터 판독한 가능 저장 매체(620)는 메모리(랜덤 액세스 메모리와 같은 휘발성 메모리, 비휘발성 메모리, 또는 이들의 적절한 조합), 하나 이상의 자기 디스크 저장 기기들, 광학 디스크 저장 기기들, 플래시 메모리 기기들, 그 밖에 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)에 의해 액세스되고 원하는 정보를 저장할 수 있는 다른 형태의 저장 매체, 또는 이들의 적합한 조합일 수 있다.

[0145] 통신 버스(660)는 프로세서(610), 컴퓨터 판독 가능한 저장 매체(620)를 포함하여 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)의 다른 다양한 컴포넌트들을 상호 연결한다.

[0146] 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)는 또한 하나 이상의 입출력 장치(미도시)를 위한 인터페이스를 제공하는 하나 이상의 입출력 인터페이스(640) 및 하나 이상의 통신 인터페이스(650)를 포함할 수 있다. 입출력 인터페이스(640) 및 통신 인터페이스(650)는 통신 버스(660)에 연결된다. 입출력 장치(미도시)는 입출력 인터페이스(640)를 통해 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)의 다른 컴포넌트들에 연결될 수 있다. 예시적인 입출력 장치는 포인팅 장치(마우스 또는 트랙패드 등), 키보드, 터치 입력 장치(터치패드 또는 터치스크린 등), 음성 또는 소리 입력 장치, 다양한 종류의 센서 장치 및/또는 촬영 장치와 같은 입력 장치, 및/또는 디스플레이 장치, 프린터, 스피커 및/또는 네트워크 카드와 같은 출력 장치를 포함할 수 있다. 예시적인 입출력 장치(미도시)는 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)를 구성하는 일 컴포넌트로서 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)의 내부에 포함될 수도 있고, 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치(10)와는 구별되는 별개의 장치로 컴퓨팅 기기와 연결될 수도 있다.

[0147] 본 실시예들에 따른 동작은 다양한 컴퓨터 수단을 통하여 수행될 수 있는 프로그램 명령 형태로 구현되어 컴퓨

터 판독 가능한 매체에 기록될 수 있다. 컴퓨터 판독 가능한 매체는 실행을 위해 프로세서에 명령어를 제공하는 데 참여한 임의의 매체를 나타낸다. 컴퓨터 판독 가능한 매체는 프로그램 명령, 데이터 파일, 데이터 구조 또는 이들의 조합을 포함할 수 있다. 예를 들면, 자기 매체, 광기록 매체, 메모리 등이 있을 수 있다. 컴퓨터 프로그램은 네트워크로 연결된 컴퓨터 시스템 상에 분산되어 분산 방식으로 컴퓨터가 읽을 수 있는 코드가 저장되고 실행될 수도 있다. 본 실시예를 구현하기 위한 기능적인(Functional) 프로그램, 코드, 및 코드 세그먼트들은 본 실시예가 속하는 기술분야의 프로그래머들에 의해 용이하게 추론될 수 있을 것이다.

[0148] 본 실시예들은 본 실시예의 기술 사상을 설명하기 위한 것이고, 이러한 실시예에 의하여 본 실시예의 기술 사상의 범위가 한정되는 것은 아니다. 본 실시예의 보호 범위는 아래의 청구범위에 의하여 해석되어야 하며, 그와 동등한 범위 내에 있는 모든 기술 사상은 본 실시예의 권리범위에 포함되는 것으로 해석되어야 할 것이다.

부호의 설명

[0149] 10: 기호 실행을 사용하는 소프트웨어 테스트 입력 생성 장치

100: 동적 오염 분석기

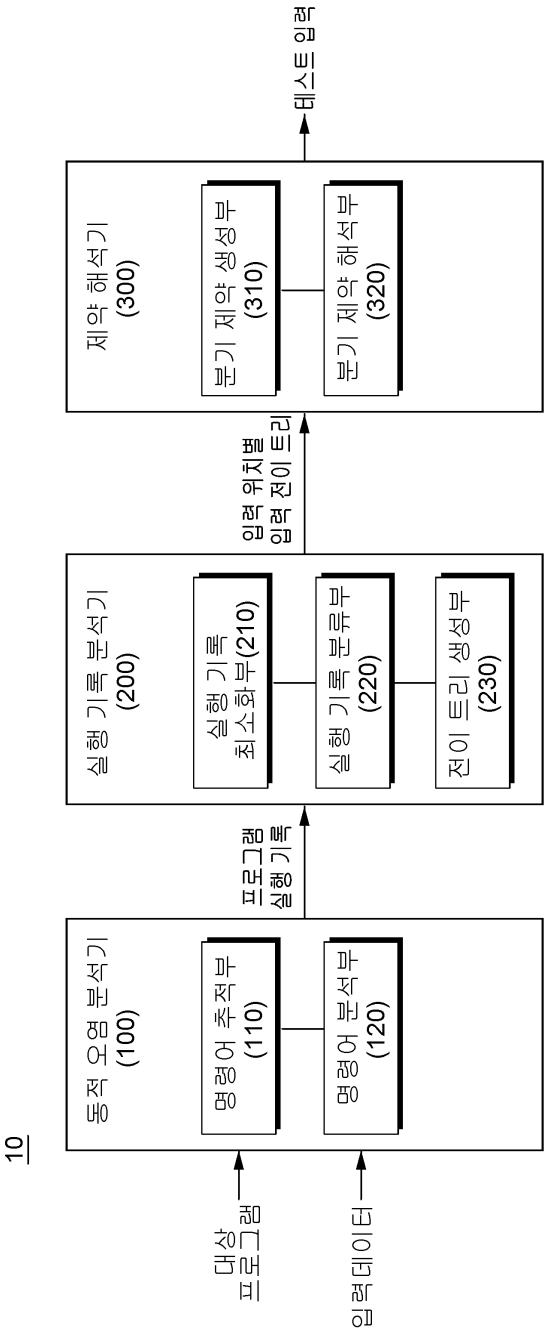
200: 실행 기록 분석기

300: 제약 해석기

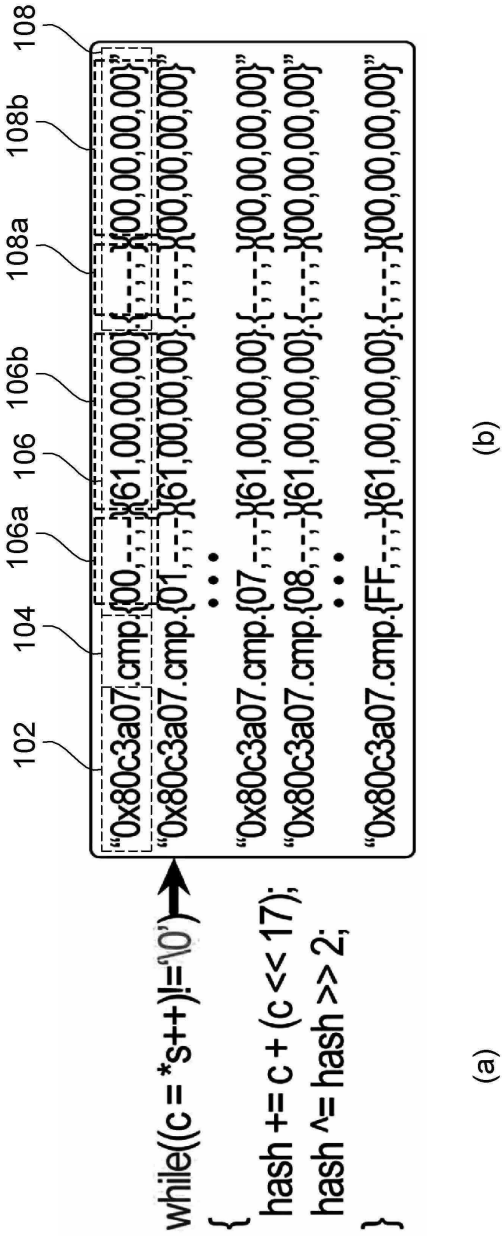
도면

도면1

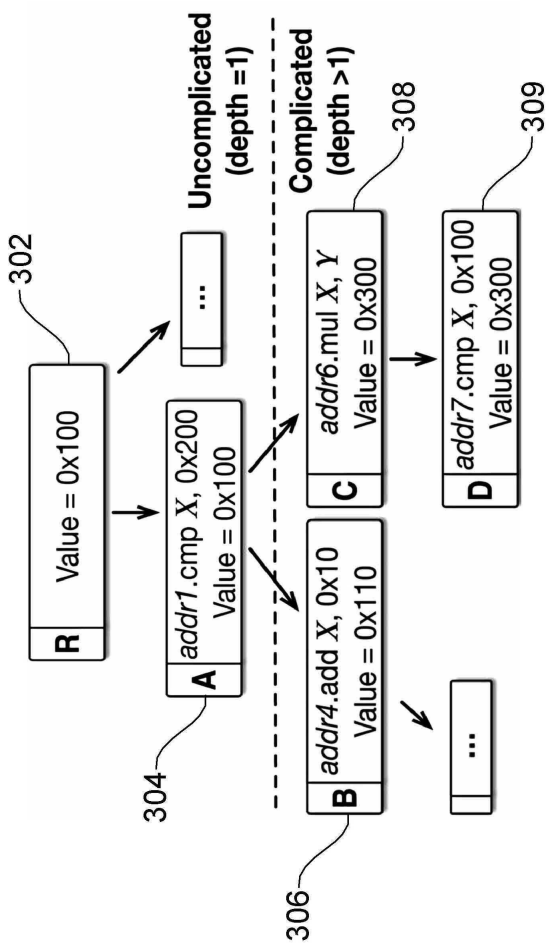
도면2


10
동적 오염 분석기
(100)
명령어 추적부
(110)
명령어 분석부
(120)
대상
프로그램
입력데이터
프로그램
실행 기록
실행 기록
분석기
(200)
실행 기록
최소화부(210)
실행 기록 분류부
(220)
전이 트리 생성부
(230)
입력 위치별
입력 전이 트리
제약 해석기
(300)
분기 제약 생성부
(310)
분기 제약 해석부
(320)
테스트 입력

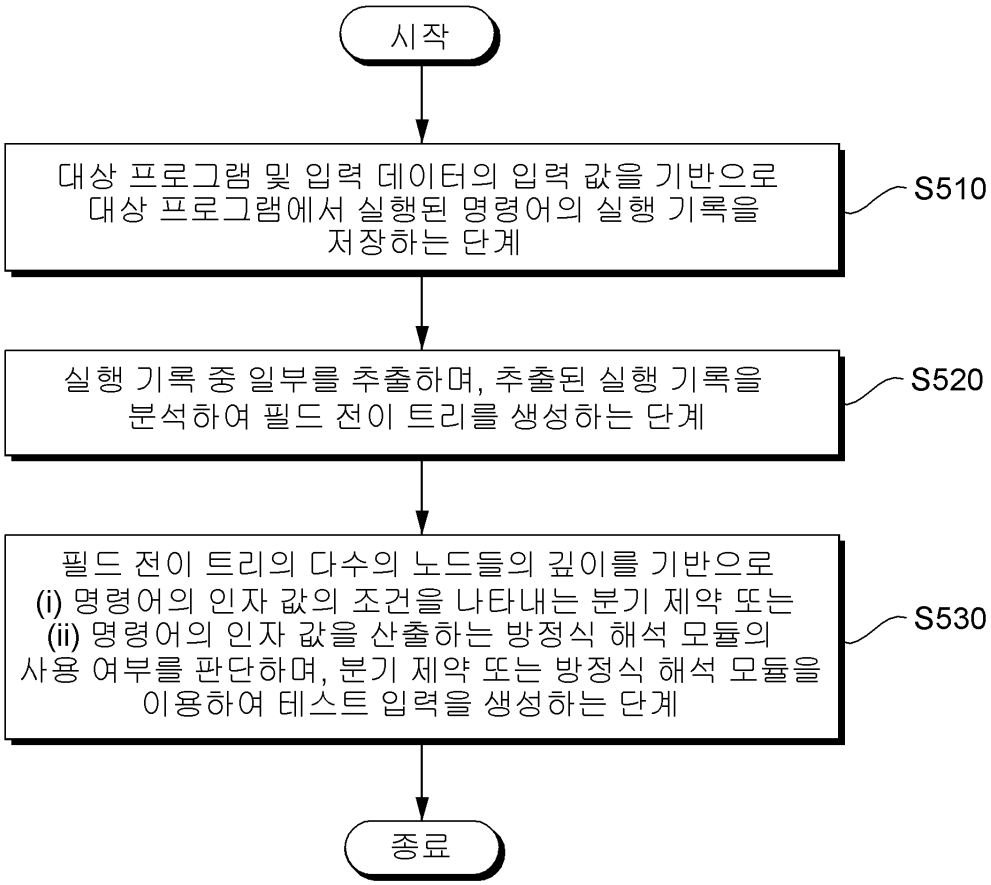
도면3


102
104
106a
106
106b
108a
108b
108
while((c = *s++)!='\0')
{
hash += c + (c << 17);
hash ^= hash >> 2;
}
"0x80c3a07.cmp.{00,-,-,-}{61,00,00,00}.{-,-,-,-}{00,00,00,00}"
"0x80c3a07.cmp.{01,-,-,-}{61,00,00,00}.{-,-,-,-}{00,00,00,00}"
• • •
"0x80c3a07.cmp.{07,-,-,-}{61,00,00,00}.{-,-,-,-}{00,00,00,00}"
"0x80c3a07.cmp.{08,-,-,-}{61,00,00,00}.{-,-,-,-}{00,00,00,00}"
• • •
"0x80c3a07.cmp.{FF,-,-,-}{61,00,00,00}.{-,-,-,-}{00,00,00,00}"
(a)
(b)

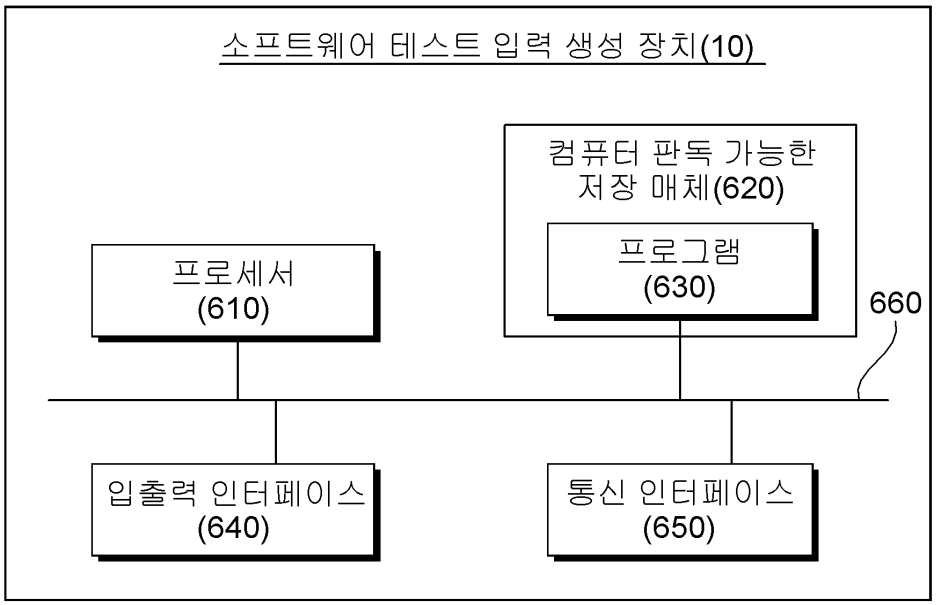
도면4


Uncomplicated
(depth =1)
Complicated
(depth >1)
302
R
Value = 0x100
A
addr1.cmp X, 0x200
Value = 0x100
304
...
B
addr4.add X, 0x10
Value = 0x110
306
C
addr6.mul X, Y
Value = 0x300
308
D
addr7.cmp X, 0x100
Value = 0x300
309
...

도면5


시작
대상 프로그램 및 입력 데이터의 입력 값을 기반으로 대상 프로그램에서 실행된 명령어의 실행 기록을 저장하는 단계
S510
실행 기록 중 일부를 추출하며, 추출된 실행 기록을 분석하여 필드 전이 트리를 생성하는 단계
S520
필드 전이 트리의 다수의 노드들의 깊이를 기반으로 (i) 명령어의 인자 값의 조건을 나타내는 분기 제약 또는 (ii) 명령어의 인자 값을 산출하는 방정식 해석 모듈의 사용 여부를 판단하며, 분기 제약 또는 방정식 해석 모듈을 이용하여 테스트 입력을 생성하는 단계
S530
종료

도면6


소프트웨어 테스트 입력 생성 장치(10)
컴퓨터 판독 가능한 저장 매체(620)
프로그램 (630)
프로세서 (610)
660
입출력 인터페이스 (640)
통신 인터페이스 (650)