



(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2024-0028122
(43) 공개일자 2024년03월05일

(51) 국제특허분류(Int. Cl.)
G06F 11/36 (2006.01) G06F 8/41 (2018.01)
(52) CPC특허분류
G06F 11/3664 (2013.01)
G06F 11/3604 (2013.01)
(21) 출원번호 10-2022-0106241
(22) 출원일자 2022년08월24일
심사청구일자 2022년08월24일

(71) 출원인
울산과학기술원
울산광역시 울주군 언양읍 유니스트길 50
(72) 발명자
이주용
울산광역시 울주군 언양읍 유니스트길 50
(74) 대리인
특허법인(유한)아이시스

전체 청구항 수 : 총 8 항

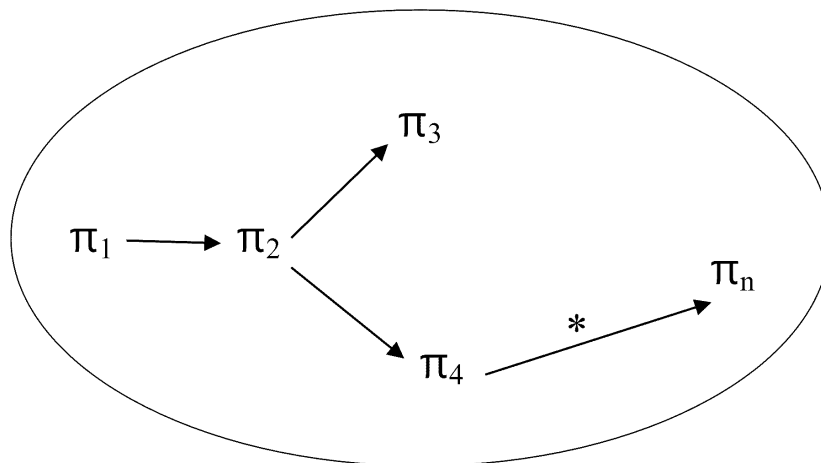
(54) 발명의 명칭 탐색 기법을 이용한 제약 조건 기반의 프로그램 수정 방법 및 장치

(57) 요약

탐색 기법을 이용한 제약 조건 기반의 프로그램 수정 방법은 분석장치가 버그 수정 대상인 소스 코드를 입력받는 단계, 상기 분석장치가 스펙트럼 기반 결함 위치 식별 기법을 이용하여 상기 소스 코드에서 오류가 의심되는 후보 표현식들을 선별하는 단계, 상기 분석장치가 상기 후보 표현식들을 기준으로 일정한 테스트를 통과한 후보 경로를 선별하는 단계 및 상기 분석장치가 상기 후보 경로를 기준으로 패치(patch)를 생성하는 단계를 포함한다.

대표도 - 도2

[실행 경로 공간]



(52) CPC특허분류

G06F 11/3624 (2013.01)

G06F 11/3688 (2013.01)

G06F 11/3692 (2013.01)

G06F 8/4432 (2013.01)

이 발명을 지원한 국가연구개발사업

과제고유번호	1711167701
과제번호	2021R1A2C1009819
부처명	과학기술정보통신부
과제관리(전문)기관명	한국연구재단
연구사업명	개인기초연구(과기정통부)
연구과제명	단일 테스트 자동 디버깅을 위한 기반 다지기
기 여 율	1/3
과제수행기관명	울산과학기술원
연구기간	2022.03.01 ~ 2023.02.28

이 발명을 지원한 국가연구개발사업

과제고유번호	1711152646
과제번호	2021-0-01001-002
부처명	과학기술정보통신부
과제관리(전문)기관명	정보통신기획평가원
연구사업명	SW컴퓨팅산업원천기술개발
연구과제명	코드 분석과 오류 마이닝이 결합된 SW 오류 자동 수정 기술 개발
기 여 율	1/3
과제수행기관명	슈어소프트테크(주)
연구기간	2022.01.01 ~ 2022.12.31

이 발명을 지원한 국가연구개발사업

과제고유번호	1711118834
과제번호	2020R1F1A1066462
부처명	과학기술정보통신부
과제관리(전문)기관명	한국연구재단
연구사업명	개인기초연구(과기정통부)(R&D)
연구과제명	자동 프로그램 수정의 속도 향상 기법 연구
기 여 율	1/3
과제수행기관명	울산과학기술원
연구기간	2020.06.01 ~ 2021.05.31

명세서

청구범위

청구항 1

분석장치가 버그 수정 대상인 소스 코드를 입력받는 단계;

상기 분석장치가 스펙트럼 기반 결함 위치 식별 기법을 이용하여 상기 소스 코드에서 오류가 의심되는 후보 표현식들을 선별하는 단계;

상기 분석장치가 상기 후보 표현식들을 기준으로 일정한 테스트를 통과한 후보 경로를 선별하는 단계; 및

상기 분석장치가 상기 후보 경로를 기준으로 패치(patch)를 생성하는 단계를 포함하고,

상기 후보 경로를 선별하는 단계는 상기 분석장치가 탐색 기반의 샘플링 기법으로 탐색된 경로의 비용을 기준으로 비용이 더 낮은 경로를 찾는 과정을 반복하되, 상기 비용은 분석 대상인 경로가 출력하는 값과 목표하는 출력값의 차이로 결정되는 탐색 기법을 이용한 제약 조건 기반의 프로그램 수정 방법.

청구항 2

제1항에 있어서,

상기 비용이 더 낮은 경로를 찾는 과정은 상기 분석장치가 MCMC(Markov Chain Monte Carlo) 샘플링 기법을 이용하여 현재 실행 경로를 새로운 경로로 변경하고, 상기 새로운 경로의 비용이 상기 현재 실행 경로의 비용보다 적은 경우 상기 새로운 경로를 선택하는 탐색 기법을 이용한 제약 조건 기반의 프로그램 수정 방법.

청구항 3

제1항에 있어서,

상기 분석 장치는 상기 테스트를 통과한 후보 경로가 출력하는 값이 상기 목표하는 출력값과 동일한 경우 경로 탐색을 종료하는 탐색 기법을 이용한 제약 조건 기반의 프로그램 수정 방법.

청구항 4

제1항에 있어서,

상기 분석장치는 상기 소스 코드가 최초 수행되면서 획득한 상기 후보 표현식들에 대한 실행값을 나타내는 참조 비트 벡터를 획득하는 단계; 및

상기 분석장치가 상기 테스트를 통과하는 조건으로 상기 후보 경로에서 상기 후보 표현식들에 대한 실행값을 나타내는 타깃 비트 벡터가 상기 참조 비트 벡터와 가장 차이가 적도록 적어도 하나의 비트를 수정하는 후처리 단계를 더 포함하고,

상기 분석장치는 상기 후처리 단계가 수행된 상기 후보 경로를 기준으로 상기 패치를 생성하는 탐색 기법을 이용한 제약 조건 기반의 프로그램 수정 방법.

청구항 5

버그 수정 대상인 소스 코드를 입력받는 입력장치;

스펙트럼 기반 결함 위치 식별 프로그램, 탐색 기반의 샘플링 프로그램 및 패치 합성 프로그램을 저장하는 저장장치; 및

상기 스펙트럼 기반 결함 위치 식별 프로그램을 이용하여 상기 소스 코드에서 오류가 의심되는 후보 표현식들을 선별하고, 상기 후보 표현식들을 기준으로 일정한 테스트를 통과한 후보 경로를 선별하고, 상기 패치 합성 프로그램을 이용하여 상기 후보 경로를 기준으로 패치를 생성하는 연산장치를 포함하되,

상기 연산장치는 상기 탐색 기반의 샘플링 프로그램을 이용하여 경로의 비용을 기준으로 비용이 더 낮은

경로를 찾는 과정을 반복하면서 상기 후보 경로를 탐색하고, 상기 비용은 분석 대상인 경로가 출력하는 값과 목표하는 출력값의 차이로 결정되는 제약 조건 기반의 프로그램 수정을 실행하는 분석 장치.

청구항 6

제5항에 있어서,

상기 비용이 더 낮은 경로를 찾는 과정은 상기 연산장치가 MCMC(Markov Chain Monte Carlo) 샘플링 기법을 이용하여 현재 실행 경로를 새로운 경로로 변경하고, 상기 새로운 경로의 비용이 상기 현재 실행 경로의 비용보다 적은 경우 상기 새로운 경로를 선택하는 제약 조건 기반의 프로그램 수정을 실행하는 분석 장치.

청구항 7

제5항에 있어서,

상기 연산장치는 상기 테스트를 통과한 후보 경로가 출력하는 값이 상기 목표하는 출력값과 동일한 경우 경로 탐색을 종료하는 제약 조건 기반의 프로그램 수정을 실행하는 분석 장치.

청구항 8

제5항에 있어서,

상기 연산장치는 상기 소스 코드가 최초 수행되면서 획득한 상기 후보 표현식들에 대한 실행값을 나타내는 참조 비트 벡터를 생성하고, 상기 테스트를 통과하는 조건으로 상기 후보 경로에서 상기 후보 표현식들에 대한 실행값을 나타내는 타깃 비트 벡터가 상기 참조 비트 벡터와 가장 차이가 적도록 적어도 하나의 비트를 수정하는 후처리 수행하고, 상기 후처리 단계가 수행된 상기 후보 경로를 기준으로 상기 패치를 생성하는 제약 조건 기반의 프로그램 수정을 실행하는 분석 장치.

발명의 설명

기술 분야

[0001] 이하 설명하는 기술은 제약 조건 기반의 자동 프로그램 수정 기법이다.

배경 기술

[0002] 자동 프로그램 수정(automatic program repair) 분야에서 프로그램 버그(bug)를 수정하는 기법으로 다양한 방법론이 연구되고 있다. 자동 프로그램 수정은 대부분 G&V(generate-and-validate) 접근법 또는 제약 조건(constraint-based) 접근법으로 구분할 수 있다.

선행기술문헌

비특허문헌

[0003] (비특허문헌 0001) Angelix: Scalable Multiline Program Patch Synthesis via Symbolic Analysis, Sergey Mechtaev, Jooyong Yi and Abhik Roychoudhury, ACM/IEEE International Conference on Software Engineering (ICSE), 2016.

발명의 내용

해결하려는 과제

[0004] 모바일 환경 발전과 함께 다양한 애플리케이션이 제공되고 있다. 실시간 서비스가 보장되어야 하는 서비스 환경에서 프로그램은 빠른 시간내에 버그 수정이 필요하다. 다만, 종래 제약 조건 기반 자동 프로그램 수정 기법은 정확한 버그 수정에 초점이 있었지 버그 수정의 런타임(runtime)을 최소화하기 위한 방향으로 연구되지 않았다.

[0005] 이하 설명하는 기술은 프로그램 버그 수정 시간에 초점이 있는 제약 조건 기반 자동 프로그램 수정 기법을 제공하고 한다. 이하 설명하는 기술은 탐색 기반 기법을 활용하여 런타임이 적은 제약 조건 기반 프로그램 버그

수정 기법을 제공하고자 한다.

과제의 해결 수단

[0006] 탐색 기법을 이용한 제약 조건 기반의 프로그램 수정 방법은 분석장치가 버그 수정 대상인 소스 코드를 입력받는 단계, 상기 분석장치가 스펙트럼 기반 결함 위치 식별 기법을 이용하여 상기 소스 코드에서 오류가 의심되는 후보 표현식들을 선별하는 단계, 상기 분석장치가 상기 후보 표현식들을 기준으로 일정한 테스트를 통과한 후보 경로를 선별하는 단계 및 상기 분석장치가 상기 후보 경로를 기준으로 패치(patch)를 생성하는 단계를 포함한다.

[0007] 제약 조건 기반의 프로그램 수정을 실행하는 분석 장치는 버그 수정 대상인 소스 코드를 입력받는 입력장치, 스펙트럼 기반 결함 위치 식별 프로그램, 탐색 기반의 샘플링 프로그램 및 패치 합성 프로그램을 저장하는 저장장치 및 상기 스펙트럼 기반 결함 위치 식별 프로그램을 이용하여 상기 소스 코드에서 오류가 의심되는 후보 표현식들을 선별하고, 상기 후보 표현식들을 기준으로 일정한 테스트를 통과한 후보 경로를 선별하고, 상기 패치 합성 프로그램을 이용하여 상기 후보 경로를 기준으로 패치를 생성하는 연산장치를 포함한다.

발명의 효과

[0008] 이하 설명하는 기술은 탐색 기반 기법을 활용하여 버그 수정에 소요되는 시간이 줄어든다. 따라서, 이하 설명하는 기술은 빠른 시간에 버그 수정이 필요한 동적인 애플리케이션의 관리에 유용하다.

도면의 간단한 설명

[0009] 도 1은 제약 조건 기반의 자동 프로그램 수정 기법에 대한 예이다.

도 2는 최적 경로 탐색에 대한 예이다.

도 3은 현재 설정과 다음 설정의 개수가 다른 경우에 대한 예이다.

도 4는 제약 조건 기반의 자동 프로그램 수정을 수행하는 분석장치에 대한 예이다.

발명을 실시하기 위한 구체적인 내용

[0010] 이하 설명하는 기술은 다양한 변경을 가할 수 있고 여러 가지 실시예를 가질 수 있는 바, 특정 실시예들을 도면에 예시하고 상세하게 설명하고자 한다. 그러나, 이는 이하 설명하는 기술을 특정한 실시 형태에 대해 한정하려는 것이 아니며, 이하 설명하는 기술의 사상 및 기술 범위에 포함되는 모든 변경, 균등물 내지 대체물을 포함하는 것으로 이해되어야 한다.

[0011] 제1, 제2, A, B 등의 용어는 다양한 구성요소들을 설명하는데 사용될 수 있지만, 해당 구성요소들은 상기 용어들에 의해 한정되지는 않으며, 단지 하나의 구성요소를 다른 구성요소로부터 구별하는 목적으로만 사용된다. 예를 들어, 이하 설명하는 기술의 권리 범위를 벗어나지 않으면서 제1 구성요소는 제2 구성요소로 명명될 수 있고, 유사하게 제2 구성요소도 제1 구성요소로 명명될 수 있다. 및/또는 이라는 용어는 복수의 관련된 기재된 항목들의 조합 또는 복수의 관련된 기재된 항목들 중의 어느 항목을 포함한다.

[0012] 본 명세서에서 사용되는 용어에서 단수의 표현은 문맥상 명백하게 다르게 해석되지 않는 한 복수의 표현을 포함하는 것으로 이해되어야 하고, "포함한다" 등의 용어는 설명된 특징, 개수, 단계, 동작, 구성요소, 부분품 또는 이들을 조합한 것이 존재함을 의미하는 것이지, 하나 또는 그 이상의 다른 특징들이나 개수, 단계 동작 구성요소, 부분품 또는 이들을 조합한 것들의 존재 또는 부가 가능성을 배제하지 않는 것으로 이해되어야 한다.

[0013] 도면에 대한 상세한 설명을 하기에 앞서, 본 명세서에서의 구성부들에 대한 구분은 각 구성부가 담당하는 주기능 별로 구분한 것에 불과함을 명확히 하고자 한다. 즉, 이하에서 설명할 2개 이상의 구성부가 하나의 구성부로 합쳐지거나 또는 하나의 구성부가 보다 세분화된 기능별로 2개 이상으로 분화되어 구비될 수도 있다. 그리고 이하에서 설명할 구성부 각각은 자신이 담당하는 주기능 이외에도 다른 구성부가 담당하는 기능 중 일부 또는 전부의 기능을 추가적으로 수행할 수도 있으며, 구성부 각각이 담당하는 주기능 중 일부 기능이 다른 구성부에 의해 전담되어 수행될 수도 있음은 물론이다.

[0014] 또, 방법 또는 동작 방법을 수행함에 있어서, 상기 방법을 이루는 각 과정들은 문맥상 명백하게 특정 순서를 기재하지 않은 이상 명기된 순서와 다르게 일어날 수 있다. 즉, 각 과정들은 명기된 순서와 동일하게 일어날 수도

있고 실질적으로 동시에 수행될 수도 있으며 반대의 순서대로 수행될 수도 있다.

- [0016] 이하 설명하는 기술은 제약 조건 기반의 자동 프로그램 수정 기법이다.
- [0017] 제약 조건 기반의 기법은 프로그램 오류 수정을 위하여 수정 제약 조건 ϕ 을 먼저 추출하고 해당 조건을 만족하는 패치(patch)를 합성한다. 수정 제약 조건 ϕ 은 패치를 위한 정보 내지 패치 명세(specification)라고 할 수 있다. 패치는 반복적인 컴파일링 및 테스트 과정으로 결정되는 것이 아니라 정확한 일련의 구축 과정으로 합성된다. 따라서, 제약 조건 기반의 기법이 이론적으로 G&V 접근법에 비하여 적은 수행시간으로 오류를 수정해야 한다. 그러나, 종래 제약 조건 기반의 기법은 수행 시간이라는 측면에서 최적화되지 못한 측면이 있다.
- [0018] 최신의 제약 조건 기반의 기법은 동일 연구자가 연구했던 ANGELIX(Angelix: Scalable Multiline Program Patch Synthesis via Symbolic Analysis, Sergey Mechtaev, Jooyong Yi and Abhik Roychoudhury, ACM/IEEE International Conference on Software Engineering (ICSE), 2016.) 기법이 있다. ANGELIX 기법은 심볼릭 실행(symbolic execution)으로 수정 제약 조건을 추출한다. 심볼릭 실행은 프로그램에서 입력값에 대한 실행 경로를 분석하기 위한 기법으로 구체적인 입력값 대신 기호로 표현된 값을 사용한다. ANGELIX 기법은 에러가 의심되는 표현식(suspicious expressions)을 심볼 α 로 대체하고, 심볼릭 실행을 사용하여 테스트를 통과하는 경로를 찾기 위하여 다양한 실행 경로를 수행한다. 심볼릭 실행을 통해 테스트를 통과하는 경로를 후보 경로라고 명명한다. 발견된 후보 경로를 따르는 제한 조건 α 는 패치 명세로 추출된다. 후보 경로를 찾는 심볼릭 실행 툴은 사전 정보 없이 다양한 심볼릭 실행 경로를 시도한다. 또한, ANGELIX 기법은 다수의 후보 경로들을 추출한다. 결국 ANGELIX 기법은 사전 정보 없이 후보 경로를 찾아가고, 다수의 후보 경로들을 찾으면서 오류 수정 시간이 많이 소요되는 단점이 있다.
- [0019] 이하 자동 프로그램 수정을 수행하는 장치를 분석장치라고 명명한다. 분석장치는 일정한 데이터 처리가 가능한 다양한 장치로 구현될 수 있다. 예컨대, 분석장치는 PC, 네트워크상의 서버, 스마트 기기, 전용 프로그램이 임베딩된 칩셋 등으로 구현될 수 있다.
- [0021] 도 1은 제약 조건 기반의 자동 프로그램 수정 기법에 대한 예이다. 분석장치는 먼저 의심스러운 표현식들을 추출한다. 분석장치는 일반적으로 스펙트럼 기반 결함 위치 식별(Spectrum-based Fault Localization, SBFL)을 이용하여 의심스러운 표현식들을 획득할 수 있다. 의심스러운 표현식은 분석 대상인 후보 표현식이라고 명명할 수도 있다. SBFL은 테스트 과정(test suite)에서 문장(statement) 및 표현식(expression)과 같은 프로그램 요소들에 대한 의심 점수를 연산한다. 도 1은 표현식 " $x > y$ "에 에러가 있다고 가정한다.
- [0022] 분석장치는 후보 표현식을 심볼릭 변수(도 1에서는 α)로 대체하여 심볼릭 실행을 한다. 분석장치는 이 과정을 통해 주어진 테스트를 통과할 수 있는 패치 명세(patch specification)를 추론할 수 있다. 도 1의 표는 패치 명세에 대한 예이다. 도 1에서 α 가 true, true, false 및 false를 갖는 경우 테스트가 성공이라고 가정한다. 즉, 시퀀스 [true, true, false, false]가 후보 경로를 구성한다고 가정한다. 도 1의 테이블은 후보 경로에서 각 변수들의 값을 나타낸다. 예컨대, α 가 처음 주어질 때 변수 x 및 y 는 각각 3과 2의 값을 갖는다. 이는 x 및 y 가 각각 3과 2의 값을 가질때 패치된(patched) 새로운 표현식이 true를 리턴한다는 의미이다.
- [0023] 마지막으로 분석장치는 프로그램 합성 기술을 사용하여 주어진 패치 명세를 만족하는 패치를 생성한다. 도 1에서 합성된 표현식 $x \geq y$ 은 패치 명세 테이블의 4개 줄(row)을 만족한다. 프로그램 합성 기술은 구성 기반 프로그램 합성(component-based program synthesis), 구문 유도 합성(syntax-guided synthesis) 등이 있다.
- [0024] 일반적인 패치 명세 추론 과정에 대하여 설명한다. 아래 표 1은 패치 명세를 생성하는 알고리즘 1(Algorithm 1)에 대한 예이다. 알고리즘 1에서 입력데이터는 프로그램 P , 테스트 케이스(I, O_e) 및 후보 표현식 세트 E 이다. 심볼릭 실행 과정에서 어느 후보 표현식 e 는 새로운 표현식으로 대체된다(라인 6). O_a 및 O_e 는 각각 실제 출력값과 기대하는 출력값이다. $O_a = O_e$ 를 만족하면, 분석장치는 경로 조건 pc 를 결정 경로 조건(R)에 추가한다. 분석장치는 증강된 경로 조건 R 의 만족 여부를 검토하면서 기대하는 출력값을 산출하는 후보 경로를 결정한다(라인 8~12). 분석장치는 모든 경로들을 탐색할 때까지 또는 최대 경로들의 갯수($\max_k$)가 충족될때 까지 후보 경로를 찾는 과정을 반복한다. 알고리즘 1에서 출력값은 후보 경로 세트 A 이다.

표 1

Algorithm 1

Input: program P
Input: test case (I, O_e) // I : input, O_e : expected output
Input: a set of suspicious expressions E
Output: angelic forest A

```

1:  $A \leftarrow \emptyset$ ;  $k \leftarrow 0$ 
2: for  $e \in E$  do
3:   while there is an unexplored path  $\wedge k++ < \max_k$  do
4:     /* Using a symbolic execution, extract */
5:     * an actual output  $O_a$  and a path condition  $pc$  */
6:      $pc, O_a \leftarrow \text{CONTROLLED\_SYMEXE}(P, I, e)$ 
7:      $R \leftarrow pc \wedge O_a = O_e$ 
8:     if  $R$  is satisfiable then
9:       /* Extract a model using a constraint solver */
10:       $M \leftarrow \text{GetModel}(R)$ 
11:       $A \leftarrow A \cup \text{EXTRACT\_ANGELIC\_PATH}(M)$ 
12:     end if
13:   end while
14: if  $A \neq \emptyset$  then
15:   break
16: end if
17: end for
18: return  $A$ 

```

[0025]

[0027] 후보 경로 세트 A 는 테스트를 통과한 각 후보 경로들로 구성된다. 후보 경로 세트 A 는 후보 경로들에 대한 $\{e : \{[(\sigma_i, v_i)]_{i=1}^{L(j)}\}_{j=1}^n\}$ 의 값으로 구성된다. (σ, v) 는 최적 값 v 및 해당 값의 환경 σ 의 쌍을 나타낸다. σ 는 위치 e 에서 변수들 및 v 가 획득된 시간에서 해당 변수의 값으로 구성된다. $[(\sigma_i, v_i)]_{i=1}^{L(j)}$ 는 $L(j)$ 길이의 시퀀스 쌍을 의미한다. 여기서 $L(j)$ 는 j 번째 후보 경로의 길이, n 은 A 에서 후보 경로의 전체 개수를 의미한다.

[0028]

아래 표 2는 후보 경로 세트에 대한 예이다.

표 2

```

{T:
  [L: [{(x=3, y=2}, true), {(x=3, y=3}, true),
        {(x=2, y=3}, false), {(x=1, y=4}, false)],
        [{(x=4, y=2}, true), {(x=6, y=3}, false),
        {(x=2, y=3}, false)}]]
  ]
}

```

[0030]

[0031] T 및 L 은 각각 고려한 테스트 및 후보 표현식(도 1의 $x > y$)의 위치를 의미한다. 표 2의 후보 경로 세트는 2개의 후보 경로들로 구성된다. 하나의 후보 경로는 $[true, true, false, false]$ 시퀀스이고, 다른 하나의 후보 경로는 $[true, false, false]$ 시퀀스이다. 여기서 후보 경로는 테스트 T 를 통과하는 경로를 말한다. 분석장치는 후보 경로 세트 중 어떤 후보 경로가 정답인지 알지 못하여 가능한 모든 후보 경로들에 대하여 패치를 생성할 수 있다. 패치를 생성할 때에 기존 프로그램에 최소한의 영향을 끼치는 패치를 선택한다.

[0033]

전술한 일반적인 패치 명세 추론은 복수의 후보 경로를 수집하기 위해 많은 시간 자원을 사용한다. 시간 효율적

으로 패치 명세 추론을 하기 위해 분석 장치는 복수의 후보 경로를 수집하는 대신에 기존 프로그램에 최소한의 영향을 끼치는 후보 경로를 직접 탐색한다. 이를 위해 분석장치는 MCMC(Markov Chain Monte Carlo) 샘플링 기법을 이용하여 실행 경로 탐색을 위한 비용을 평가한다. 실행 경로의 비용은 비용 함수를 통해 결정될 수 있다. 비용 함수는 다양한 함수 중 어느 하나가 이용될 수 있다. 예컨대, 기대하는 출력값이 문자열(string)과 같은 형태라면 비용은 기대하는 출력값과 실제 출력값 사이의 편집 거리(edit distance)일 수 있다. 편집 거리는 문자열의 유사도를 평가하는 알고리즘에서 활용되는 값이다.

[0034] 분석장치는 단순한 심볼릭 실행을 수행하지 않고, MCMC 기법을 활용하여 후보 경로를 찾을 수 있다. MCMC 기법은 높은 확률을 갖는 샘플을 선별하는 기법이다. 샘플은 비트 벡터 형태로 표현되는 실행 경로이다. 예컨대, 표 2에서 첫번째 후보 경로는 [true, true, false, false]라는 1100으로 표현될 수 있다.

[0035] 연구자는 다양한 MCMC 샘플링 기법들 중 메트로폴리스 헤이스팅스 알고리즘(Metropolis-Hastings algorithm)을 사용하였다. 해당 알고리즘은 현재 샘플 S를 변형하면서 새로운 샘플 S*를 반복적으로 생성한다. 따라서 분석장치는 비트-벡터로 표현되는 현재 실행 경로를 변형하여 새로운 실행 경로를 생성한다.

[0037] MCMC 샘플링 기법을 통해 후보 경로가 찾아지면, 분석장치는 찾아진 후보 경로 주변 경로들을 확인하여 원시 프로그램(original program)에서 관찰된 경로와 최대한 유사한 제 3의 후보 경로를 선별한다. 이는 기존 프로그램에 최소한의 영향을 끼치는 패치를 생성하기 위함이다.

[0038] 분석장치는 아래 두 가지 조건을 만족하는 비트 벡터 형태의 후보 경로 $\vec{b}$ 선별할 수 있다. (i) 먼저 분석장치는 주어진 테스트를 통과하는 $\vec{b}$ 를 선택한다. (ii) 그리고 동시에 분석장치는 원본 실행 경로에 대응하는 비트 벡터에 가장 가까운 $\vec{b}$ 를 선택한다. 이 과정에서 연구자는 델타 디버깅(delta debugging)을 적용하였다. 아래 표 3은 델타 디버깅을 적용한 알고리즘 ddmin(Δ, n)에 대한 예이다.

표 3

Algorithm 2 ddmin(Δ, n)

Input: a list of elements Δ

▷ When ddmin is initially called, $\Delta = |\vec{b} - \vec{b}_1|$ where $\vec{b}$ and $\vec{b}_1$ represent an angelic path and the original execution path, respectively. Note that a given test passes in an angelic path (i.e., $test(\vec{b}) = \checkmark$) while the same test fails for the original path (i.e., $test(\vec{b}_1) = \times$).

Input: a number of partitions n

▷ Δ is divided into n partitions

Output: a list of elements $\Delta_{min} \subseteq \Delta$ satisfying the following:

▷ $test(update(\vec{b}_1, \Delta_{min})) = \checkmark$

▷ $\forall \delta_i \in \Delta_{min}. test(update(\vec{b}_1, \Delta_{min} - \{\delta_i\})) = \times$

1: **while** $test(update(\vec{b}_1, \Delta_i)) = \checkmark$ **do** ▷ $\Delta = \Delta_1 \cup \dots \cup \Delta_n$

2: /* reduce to subset */

3: **return** ddmin(update($\vec{b}_1, \Delta_i$), 2)

4: **end while**

5: **while** $test(update(\vec{b}_1, \nabla_i)) = \checkmark$ **do** ▷ $\nabla_i = \Delta - \Delta_i$

6: /* reduce to complement */

7: **return** ddmin(update($\vec{b}_1, \nabla_i$), max($n-1$, 2))

8: **end while**

9: **if** $n < |\Delta|$ **then**

10: /* increase granularity */

11: **return** ddmin(Δ , min($|\Delta|$, 2n))

12: **else**

13: **return** Δ

14: **end if**

[0040]

[0042] 후보 경로 후보 $\bar{b}$ 과 원시 실행 경로 $\bar{b}_1$ 의 차이는 $\Delta = |\bar{b} - \bar{b}_1|$ 로 표현된다. 예컨대, $\bar{b}_1$ 가 1000이고 $\bar{b}$ 가 0001라면 두 벡터의 차이는 첫 번째 비트와 마지막 비트가 다르다. ddmin 알고리즘은 분할 정복법(divide and conquer)을 사용하여 두 가지 조건을 만족하는 $\Delta_{\min} \subseteq \Delta$ 를 리턴한다. 두 가지 조건은 (1) $test(update(\bar{b}_1, \Delta_{\min})) = \checkmark$ 및 (2) $\forall \delta_i \in \Delta_{\min} \cdot test(update(\bar{b}_1, \Delta_{\min} - \{\delta_i\})) = \times$ 이다. 여기서 $update(\bar{b}, \Delta)$ 는 Δ 로 $\bar{b}$ 를 업데이트하는 함수이다. 예컨대, $update(1000, \{\text{첫번째 비트를 0으로 변경, 마지막 비트를 1로 변경}\})$ 의 결과는 0001이다. $\checkmark$ 및 $\times$ 는 각각 테스트 성공과 실패를 의미한다.

[0044] 이하 설명하는 제약 조건 기반의 자동 프로그램 수정 기법은 종전 기법과 유사하게 (1) 결함 위치 식별, (2) 패치 명세 추론 및 (3) 패치 합성 과정으로 구성된다. 다만 분석장치는 초기 실행 경로 π_1 로부터 시작하는 랜덤 워크(random walk)를 통해 후보 경로 π_n 을 검색한다. 즉, 분석장치가 MCMC 알고리즘에 기반하여 후보 경로를 찾는다.

[0046] MCMC 샘플링은 현재 샘플 S 를 변환(mutation)하여 새로운 샘플 S^* 를 반복적으로 제안하는 무작위 보행(random walk)를 수행한다. 자동 프로그램 수정에서 무작위 보행은 현재 실행 경로를 무작위로 변환하여 새로운 실행 경로(다음 실행 경로)를 획득하는 과정이다.

[0047] MCMC 샘플링은 현재 선택한 다음 실행 경로 S^* 를 항상 선택하지 않는다. MCMC 샘플링은 아래 수학식 1을 만족하는 확률 $\alpha(S \rightarrow S^*)$ 을 기준으로 S^* 를 선택한다.

수학식 1

[0048]
$$\alpha(S \rightarrow S^*) = \min \left(1, \frac{p(S^*)}{p(S)} \cdot \frac{q(S|S^*)}{q(S^*|S)} \right)$$

[0049] $p(S)$ 및 $p(S^*)$ 는 각각 S 및 S^* 의 밀도 확률 함수(probability density function)이다. $q(S|S^*)$ 는 주어진 S 에 대하여 S^* 를 제안하는 조건부 확률을 말한다. 밀도 확률 함수 $p(S)$ 는 S 에 대한 임의 비용 함수 c 와 함께 아래 수학식 2와 같이 표현 가능하다.

수학식 2

[0051]
$$p(S) = \frac{1}{Z} \exp(-\beta \cdot c(S))$$

[0052] β 는 조절가능한 상수, Z 는 분포를 정규화하는 분할 함수이다. 수학식 1 및 수학식 2를 조합하면 경로 선택 확률은 아래 수학식 3과 같이 정리될 수 있다.

수학식 3

$$\alpha(S \rightarrow S^*) = \min \left(1, \exp(-\beta \cdot k) \cdot \frac{q(S|S^*)}{q(S^*|S)} \right)$$

[0054]

[0055]

$k = c(S^*) - c(S)$ 이다. $c(S^*)$ 와 $c(S)$ 는 실행 경로 S^* 와 S 를 수행하는 데에 소요되는 비용을 지칭한다. 수학식 3에서 Z 는 제거되었다. $\alpha(S \rightarrow S^*)$ 연산은 각 제안 경로에 대한 비용과 두 개의 확률 조건 $q(S^*|S)$ 및 $q(S|S^*)$ 이 필요하다.

[0057]

도 2는 후보 경로 탐색에 대한 예이다. 탐색할 다음 실행 경로는 현재 경로(최초 π_1 으로 설정)를 무작위적으로 변환하여 준비된다. 분석장치는 각 실행 경로 π_i 에 대하여 해당 경로 비용 $c(\pi_i)$ 를 측정한다. 만약 $c(\pi_i)$ 가 현재 실행 경로의 비용 (π_c)보다 적다면 분석장치는 현재 실행 경로를 π_i 로 업데이트 한다. 반대로 $c(\pi_i)$ 가 현재 실행 경로의 비용보다 많다면 분석장치는 확률 p 로 π_i 를 버린다. 나머지 $1-p$ 의 확률로 분석장치는 현재 실행 경로를 π_i 로 업데이트한다. 이는 전역최적해(global optimum) 대신 국부최적해(local optimum) 방향으로 탐색을 하는 것을 방지하기 위함이다

[0059]

확률 p 는 $1 - \alpha(\pi_c \rightarrow \pi_i)$ 로 정의될 수 있다. 여기서 $\alpha(\pi_c \rightarrow \pi_i)$ 는 π_c 에서 π_i 로 현재 실행 경로를 업데이트하는 확률을 의미한다. π_i 를 버릴 확률 p 는 ' $c(\pi_i) - c(\pi_c)$ '가 증가할 수록 증가한다. $\alpha(\pi_c \rightarrow \pi_i)$ 는 수학식 3에서 S 및 S^* 에 각각 π_c 와 π_i 를 대입하여 계산한다.

[0060]

수학식 3에서 S 및 S^* 는 크기가 동일하다고 가정하였다. 그러나, 실행 경로에 따라 고려할 후보 표현식들은 서로 다른 횟수만큼 수행될 수 있다. 즉, S 및 S^* 는 크기가 동일하지 않을 수 있다. 또한, 실행 경로를 나타내는 비트 벡터들은 일정한 확률로 변환되지 않고 변환 정도가 동적이어서 MCMC 적용에 어려움이 있을 수 있다. 이하 상기 문제를 극복하기 위한 방법을 설명한다.

[0062]

분석 장치는 모든 후보 경로를 찾지 않고 MCMC 샘플링을 하여 단일한 후보 경로를 찾고자 한다. 주어진 프로그램 위치는 조건부 표현식 e 를 포함하는데 분석장치는 프로그램 실행 과정에서 e 가 수행될때마다 프로그램에서 어떤 가지(branch) 방향이 수행되는지 모니터링한다. 분석장치는 가지 방향을 나타내는 불리언(Boolean) 값의 시퀀스를 저장한다. 분석장치는 현재 비트 벡터를 변환하여 설정 파일에 저장하고 다음 프로그램 수행 과정에서 실행 경로가 설정 파일에 저장된 가지 방향을 따르도록 한다. 이때 분석장치는 비용을 연산하여 새로운 설정 S 가 종전에 저장된 설정의 비용보다 낮은 경우에 새로운 설정을 설정 파일에 저장한다. 새로운 설정 S 가 종전에 저장된 설정의 비용이 높다면, 분석장치는 확률 p 로 S 를 버리거나 $1-p$ 의 확률로 S 를 선택한다.

[0063]

MCMC 샘플링을 사용하는 제약 조건 기반의 자동 프로그램 수정 기법은 심볼릭 실행을 사용할 필요가 없다. 분석 장치는 버그가 존재하는 프로그램에서 의심스러운 각 표현식을 래퍼 함수(wrapper function)를 사용하여 변환한다. 아래 표 4는 래퍼 함수를 사용하여 변환한 예이다. 실행 과정에서 래퍼 함수는 매칭되는 설정값에 정의된 값을 리턴한다.

표 4

```
while (wrapper(/* the original expression: */
    x > y,
    /* location of expression: */
    "10-7-10-11",
    /* names of environment variables: */
    ((char*[]){ "x", "y" }),
    /* values of environment variables: */
    ((int[]){ x, y }),
    /* the number of environment variables: */
    2)) { ... }
```

[0065]

[0067] 분석장치는 설정을 반복적으로 변환하면서 후보 경로를 찾는다. 설정은 $\{1:[0,1]^+\}$ 로 정의될 수 있다. 여기서 1은 후보 표현식의 프로그램 위치를 나타낸다. $[0,1]^+$ 는 최소한 1 길이를 갖는 비트 벡터이다. 비트 벡터에서 i 번째 비트는 위치 1의 조건부 표현식의 i 번째 실행에서 사용되어야 하는 불리언 값을 나타낸다. 아래 표 5는 설정에 대한 예이다. 표 5의 설정은 위치 10-7-10-11에 위치한 후보 표현식 e 가 순서대로 1, 1, 0 및 0을 리턴해야 하는 것을 나타낸다.

표 5

```
{"10-7-10-11": [1, 1, 0, 0]}
```

[0069]

[0071] 아래 표 6은 명세 추론 과정에 대한 알고리즘(Algorithm 3)의 예이다.

표 6

Algorithm 3 The specification inference algorithm

Input: program P
Input: test case (I, O_e) // I : input, O_e : expected output
Input: a sorted set of locations L of suspicious conditional expressions
Input: cost function Cost
Output: an angelic path π

```

1: /* Extract the original output  $O_1$  and */
2: /* the value trace of  $E$  denoted with  $\bar{S}$  */
3:  $O_1, \bar{S} \leftarrow \text{RUN}(I, L)$ 
4:  $C_1 \leftarrow \text{Cost}(O_1, O_e)$  //  $C_1$ : initial cost
5: for  $l \in L$  do
6:   /* Phase 1: Initialization */
7:    $S \leftarrow \bar{S}[l]$  //  $S$ : initial configuration for  $l$ 
8:    $O, C \leftarrow O_1, C_1$ 
9:    $N \leftarrow 0$  //  $N$ : the number of trials
10:  /* Phase 2: Search for an angelic path */
11:  while  $O \neq O_e \wedge \text{CONTINUE}(N, C)$  do
12:    /* perform MCMC sampling */
13:     $S' \leftarrow \text{PROPOSE}(S)$ 
14:     $O, S^* \leftarrow \text{RUN}(I, S')$ 
15:     $C^* \leftarrow \text{Cost}(O, O_e)$ 
16:    if  $\text{ACCEPT}(C, C^*)$  then
17:       $S, C \leftarrow S^*, C^*$ 
18:    end if
19:     $N \leftarrow N + 1$ 
20:  end while
21:  /* Phase 3: Post-processing */
22:  if  $O = O_e$  then
23:     $S_1 \leftarrow \bar{S}[l]$  // the original buggy configuration for  $l$ 
24:     $S^\dagger \leftarrow \text{REFINE}(S, S_1)$ 
25:    return  $S^\dagger$ 
26:  end if
27: end for
28: return  $\perp$  // inference failure

```

[0073]

[0074] 분석장치는 테스트 입력 I 를 갖고 수정 대상 프로그램 P 를 수행하면서 관찰되는 설정을 초기 설정으로 사용할 수 있다. 분석장치는 SBFL을 수행하여 후보 표현식의 위치를 선택할 수 있다. 분석장치는 의심 정도를 나타내는 점수 순서 내림 차순에 따라 L 을 저장할 수 있다.

[0075] 알고리즘 3의 라인 5 ~ 27은 3개의 단계로 구성된다. (1) 첫 번째 단계(line 7-9)는 현재 설정 S , 현재 출력 O , 현재 비용 C 및 시도 횟수 N 을 포함하는 변수를 초기화하는 단계이다. 변수 S , C 및 O 는 최초 프로그램을 수행한 결과로 획득되는 설정, 출력값 및 비용으로 설정된다. (2) 두 번째 단계(line 11-20)는 MCMC 샘플링을 통해 후보 경로를 탐색하는 단계이다. 분석장치는 현재 설정 S 를 변형하여 S' 를 생성한다. $\text{Propose}()$ 가 S 를 변형하는 함수를 나타낸다. 분석장치는 S' 를 갖는 프로그램을 수행하여 출력 O 를 얻으면서 비용 C^* 를 연산한다. 분석장치는 비용에 따라 현재 설정을 선택할지를 결정한다. $\text{ACCEPT}()$ 가 비용을 비교하여 현재 설정을 선택하는 함수를 의미한다. (3) 마지막 단계는 후보 경로가 발견되면 수행된다. 후보 경로는 출력하는 O 가 기대된 출력 O_e 와 동일한 경우이다. 분석장치는 획득한 후보 경로를 $\text{REFINE}()$ 을 호출하여 후처리한다(line 24). $\text{REFINE}()$ 은 전술한 델타 디버깅을 수행하여 원시 설정에 가장 가까운 후보 경로를 찾는 함수이다. 이하 각 과정을 대하여 상세하게 설명한다.

[0077] 분석장치가 MCMC 샘플링 과정에서 설정 S 를 변경하는 과정(PROPOSE 함수)을 설명한다. 주어진 S 가 $\{1:\vec{b}\}$ 로 정의

된다. 1 및 $\bar{b}$ 는 후보 표현식의 위치 및 해당 표현식의 비트 벡터이다.

- [0078] 분석장치는 $\bar{b}$ 에서 p_1 의 확률로 하나의 비트를 바꾸거나, $\bar{b}$ 중 $1-p_1$ 의 확률을 갖는 n 개 비트(들) 임의로 바꿀 수 있다. p_1 은 설정된 값을 사용할 수 있다. 연구자는 실험적으로 p_1 을 0.5로 설정하여 사용하였다. 분석장치가 다수(k 개)의 비트들을 변경하는 경우 하나의 비트를 k/n 과 같이 일정한 확률로 변경할 수 있다.
- [0079] 분석장치가 현재 설정 S 를 변환하여 다음 설정 S' 를 생성한다면, S' 의 비트 벡터 길이($|S'|$)는 S 와 동일하다. 그러나, 수정 전의 프로그램이 S' 로 실행되면 현재 의심스러운 위치가 실행되는 횟수가 $|S'|$ 의 개수와 다를 수 있다.
- [0080] 분석장치는 S' 로 프로그램을 테스트하면서 현재 의심스러운 위치 1에서 표현식 e 의 값을 추적하여 저장한다. 분석장치는 테스트 과정에서 e 가 실행될 때마다 S' 에 명세된 값들을 사용한다. 테스트 과정에서 e 가 실행되는 횟수가 $|S'|$ 보다 많다면(S' 에 명세된 값들이 모두 소모되면), 분석장치는 e 가 수행될 때마다 임의의 비트값을 리턴한다. 만약, e 가 실행되는 횟수가 $|S'|$ 보다 작다면 분석장치는 S' 에서 실제 사용된 비트값들을 추출하여 S^* 로 설정을 조정한다(line 14).
- [0081] 분석장치는 MCMC 샘플링을 위하여 각 설정의 비용을 측정한다(line 15). 비용은 후보 경로와 S^* 에 따른 실행 경로 사이의 거리로 산정될 수 있지만, 이 시점에서 분석장치는 후보 경로를 알지 못하기에 사용자가 제공하는 특정 비용 함수를 사용할 수 있다. 비용 함수는 기대되는 출력값 O_e 과 S^* 에 따른 실제 출력값 O 의 거리로 측정될 수 있다.
- [0082] 분석장치는 메트로폴리스 헤이스팅스 알고리즘을 사용하여 설정을 업데이트한다. 분석장치는 현재 비용 C 와 설정 S^* 가 사용되는 경우의 비용 C^* 를 비교한다. $C^* < C$ 라면, 분석장치는 S^* 를 새로운 설정으로 선택한다. $C^* \geq C$ 라면, 분석장치는 S^* 를 무시하고, 확률 $p < 1$ 인 현재 설정을 유지한다. $C^* \geq C$ 라면, 분석장치는 p 의 확률로 S^* 를 무시하고 $1-p$ 의 확률로 S^* 를 새로운 설정으로 선택한다. 확률 p 는 $1 - \alpha(S \rightarrow S^*)$ 로 정의되며, $\alpha(S \rightarrow S^*)$ 는 수학식 3을 통해 구한다.
- [0083] 분석장치는 수학식 3으로 표현되는 비용 기반의 메트로폴리스 헤이스팅스 알고리즘을 사용할 수 있다. 수학식 3에서 β 는 상수(연구자는 실험적으로 0.8 사용)이고, $k = \text{COST}(S^*) - \text{COST}(S)$ 이다. $\text{COST}()$ 는 비용을 산정하는 함수이다. $\alpha(S \rightarrow S^*)$ 는 현재 설정이 S 이고 다음 설정으로 S^* 를 선택할 확률이다.
- [0084] 전술한 바와 같이 S 와 S^* 의 비트 벡터 길이가 다를 수 있다. 따라서 동적인 환경에 따라 분석장치가 $q(S^*|S)$ 를 연산해야 한다. $q(S^*|S)$ 는 S 가 S^* 로 변환될 확률을 나타낸다. 먼저 분석장치는 $|S| = |S^*|$ 인 경우 $\bar{b}$ 에서 p_1 의 확률로 하나의 비트를 바꾸거나, $1-p_1$ 의 확률로 n 개의 비트를 바꾼다(단, $1 < n \leq |S|$).
- [0085] 현재 설정 S 와 후보 설정 S' 간에 k 개의 비트에서 차이가 날 때, 해당 k 개 비트들을 임의로 바꿀 수 있다. k 개의 특정 비트들을 변경할 확률은 아래 수학식 4와 같다. $|S| = 1$ 이라고 가정한다.

수학식 4

$$(l-1)^{-1} \cdot \binom{l}{k}^{-1} \cdot (1-p_1)$$

[0087]

$$(1-p_1)^{-1}$$

[0088] $(1-p_1)^{-1}$ 은 2와 1 사이의 갯수인 k 를 선택할 확률이다. $\binom{l}{k}^{-1}$ 은 1 비트들 중 k 개 특정 비트 조합을 선택할 확률이다. $(1-p_1)$ 은 하나의 비트보다 많은 비트들을 변경할 확률로 사용자가 정의하는 값이다.

[0090] S와 S'가 하나의 특정 비트만 차이가 난다면, S가 S'로 변환될 확률은 아래 수학적 식 5와 같다.

수학적 식 5

[0092] $I^{-1} \cdot p_1$

[0093] I^{-1} 은 S와 S' 사이에서 하나의 특정 비트만 변경할 확률이다. p_1 는 하나의 비트를 변경할 확률로 사용자가 정의하는 값이다.

[0095] 도 3은 현재 설정과 다음 설정의 개수가 다른 경우에 대한 예이다.

[0096] 도 3(A)는 $|S^*| > |S|$ 인 경우이다. 도 3(A)는 S, S' 및 S^* 가 각각 010, 111 및 11110인 예이다. S' 및 S^* 사이에서 공통된 비트들은 밑줄로 표시하였다. 이 경우 변환 확률은 아래 수학적 식 6과 같은 표현할 수 있다.

수학적 식 6

[0098] $q(S^*|S) = q(S'|S) \cdot (1/2)^{(|S^*| - |S|)}$

[0099] $|S| = |S'|$ 이기 때문에 $q(S'|S)$ 는 수학적 식 4 및 5로 연산될 수 있다. 한편, $(1/2)^{(|S^*| - |S|)}$ 는 수행 시간 동안 임의의 비트값 생성으로 S^* 의 후행 비트가 생성되는 확률이다. 연구자는 실험적으로 후행하는 비트 시퀀스 10을 1/4 확률로 생성하였다.

[0101] 도 3(B)는 $|S^*| < |S|$ 인 경우이다. 이는 현재 후보 표현식이 S'에 명세된 횃수보다 적게 실행되는 경우이다. 도 3(B)를 살펴보면, S'는 두 가지 경우에 따라 0100 및 0101로 생성될 수 있다. 이와 같은 경우를 고려하여 변환 확률은 아래 수학적 식 7과 같이 표현할 수 있다.

수학적 식 7

[0103] $q(S^*|S) = \sum_{S' \in \text{Ext}(S^*)} q(S'|S)$

[0104] 함수 Ext는 주어진 S^* 를 가능한 모든 설정 S' 세트에 확장한다. S'의 앞 부분 $|S^*|$ 개 비트는 S^* 와 동일하다. 그리고 $|S'| = |S|$ 이다. $\text{Ext}(010) = \{0100, 0101\}$ 이다. $q(S'|S)$ 는 수학적 식 4와 5를 이용하여 연산될 수 있다.

[0106] 한편, 수학적 식 4에서 $q(S|S^*)$ 는 $q(S^*|S)$ 와 유사하게(대칭적인 방식) 연산될 수 있다. $|S^*| = |S|$ 인 경우, $q(S|S^*)$ 는 $q(S^*|S)$ 와 동일하다. 이 경우 아래 수학적 식 8과 같이 수식을 단순하게 표현할 수 있다.

수학식 8

$$\alpha(S \rightarrow S^*) = \min(1, \exp(-\beta \cdot k))$$

나아가, 분석장치는 다음과 같은 조건의 경우 후보 경로 탐색을 중단한다. (i) 첫째로, 분석장치는 다수의 후보 경로를 찾지 않는다. 분석장치는 $0 = 0^\circ$ 인 후보 경로를 발견하면 추가적인 탐색을 하지 않는다. (ii) 둘째로, 샘플링 시도를 위한 최대 임계값 $N_{\max}$ 를 설정하고, 분석장치는 샘플링 시도 횟수 N 이 $N_{\max}$ 를 초과하면 탐색을 중단한다. (iii) 셋째로, 분석장치는 추가 탐색을 수행하는데 비용이 줄어들지 않는다면 탐색을 중단한다. 알고리즘 3의 CONTINUE()함수(line 11)는 상기 둘째 조건 및 셋째 조건이 발생하면 false를 산출한다.

알고리즘 3에 따른 명세 추론 과정에서 산출되는 후보 경로가 잘못된 경우, 오류가 있는 패치가 생성될 수 있다. 이와 같은 문제를 해결하기 위하여 분석장치는 REFINE 함수로 표현한 후처리 과정을 수행한다.

설정은 의심스러운 프로그램 위치를 나타내는 비트 벡터로 구성된다. 후보 경로를 발견한 현재 주어진 설정 S (후보 설정)가 있다면, 분석장치는 초기 프로그램의 실행 경로에서 획득가능한 초기 설정 S_1 과 S 를 비교하여 주어진 테스트를 통과하면서 가장 최소의 비트 변화만을 갖는 설정을 찾는다. 즉, 분석장치는 초기 실행 경로와 가장 유사한 후보 경로를 찾는다.

분석장치는 델타 디버깅을 사용하여 초기 설정을 기준으로 최소 변화를 갖는 비트를 탐색한다. 비트 벡터 $\vec{b}$ 를 포함하는 후보 설정 S^* 가 있다고 가정한다. S^* 는 하나의 비트라도 변경되면 에러를 유발하는 설정(1-minimal configuration)을 말한다.

후보 표현식 e 에 대한 초기 설정 S_1 및 e 에 대한 후보 설정 S^* (후보 경로를 발견한 설정)가 있다면, S^* 는 $[1 \leq i \leq L(\vec{b}) : \text{RUN}(I, S^*[e : \vec{b}^*[i] \mapsto \vec{b}_1[i]]) \neq O_e]$ 를 만족하면 1-minimal을 만족한다. RUN() 함수는 알고리즘 3에서 테스트를 수행하는 함수이다. $L(\vec{b})$ 는 비트 벡터 $\vec{b}$ 의 길이이다. $S^*[e : \vec{b}^*[i] \mapsto \vec{b}_1[i]]$ 는 $\vec{b}$ 의 i 번째 비트가 $\vec{b}_1[i]$ 로 대체되어 수정된 설정 S^* 을 의미한다.

MCMC 샘플링 기법을 사용하면 자동 프로그램 수정에서 심볼릭 실행을 사용하지 않을 수 있다. 이는 조건부 표현식의 실행에서 그 값이 true 또는 false이기 때문이다. 그러나, 프로그램 구문이 RHS(Right-Hand Side)로 표현되는 경우 표현식의 값이 작은 개수의 값으로 표현되지 못할 수 있다. 따라서, 이 경우 분석장치는 RHS로 표현되는 문장에 대해서는 심볼릭 실행을 이용할 수도 있다.

심볼릭 실행에서 입력된 심볼이 프로그램 위치 L_i 의 조건부 표현식 C_i 에 주입된다고 가정한다. 분석장치는 심볼릭 실행 과정에서 C_i 의 가지 중 선택된 가지를 기록한다. 아래 표 7은 심볼릭 실행에 대한 예이다. 표 7은 foo.c 파일의 프로그램 위치 L_i 에서 조건부 표현식을 나타낸다. 각 L_i 는 비트벡터 형태로 기록된다.

표 7

$\{ "foo.c" : \{ L_1 : [1, 1, 0, 0], L_2 : [1, 0] \} \}$
--

[0123] 연구자는 전술한 명세 추론 알고리즘을 PyMC3에서 구현하였다. 연구자는 동일인이 연구한 종래 알고리즘 (ANGELIX)와 전술한 알고리즘(MCMC 샘플링과 후처리 적용)을 성능을 비교하였다. 연구자는 5가지 유형의 애플리케이션(Libtiff, PHP, GMP, Gzip, Wireshark))에 대하여 오류를 갖는 50개의 버전을 대상으로 프로그램 오류 수정 성능을 평가하였다. 실험 결과 전술한 새로운 알고리즘이 종전 알고리즘(ANGELIX)에 비하여 평균적으로 3.5배 빠르고, 최대 23배까지 빠른것을 확인하였다. 아래 표 8은 검증 결과를 정리한 내용이다. FAngelix는 MCMC 샘플링과 후처리 적용한 제안한 알고리즘을 의미한다. version은 오류 수정 대상인 소스 프로그램을 의미한다.

표 8

Version	Tool	Patch found	Correct patch	Time (s)			Version	Tool	Patch found	Correct patch	Time (s)		
				Median	IQR	Speedup					Median	IQR	Speedup
L-09e82	Angelix	0	0	330	24	×1.45	P-01745	Angelix	10	0	916	12	×4.48
L-09e82	FAngelix	0	0	227	10		P-01745	FAngelix	10	0	205	4	
L-15632	Angelix	10	0	564	4	×1.1	P-0de2e	Angelix	0	0	6 530	9315	×6.9
L-15632	FAngelix	10	0	512	91		P-0de2e	FAngelix	0	0	947	1526	
L-2e8b2	Angelix	0	0	263	7	×1.3	P-11941	Angelix	10	0	337	21	×1.73
L-2e8b2	FAngelix	0	0	202	47		P-11941	FAngelix	10	0	195	2	
L-37133	Angelix	10	10	1 903	7	×1.4	P-14738	Angelix	0	0	676	590	×1.55
L-37133	FAngelix	10	10	1 355	65		P-14738	FAngelix	0	0	436	599	
L-3b848	Angelix	10	0	8 235	326	×2.18	P-187eb	Angelix	10	0	316	9	×1.58
L-3b848	FAngelix	10	10	3 770	106		P-187eb	FAngelix	10	0	200	3	
L-60747	Angelix	0	0	4295	7681	×14.31	P-1f499	Angelix	0	0	567	391	×1.41
L-60747	FAngelix	0	0	300	173		P-1f499	FAngelix	0	0	403	529	
L-64062	Angelix	0	0	14 400	0	×14.75	P-2a696	Angelix	0	0	2 792	1973	×2.85
L-64062	FAngelix	0	0	976	887		P-2a696	FAngelix	0	0	979	653	
L-6746b	Angelix	0	0	14 400	0	×16.1	P-53204	Angelix	0	0	1 818	623	×11.47
L-6746b	FAngelix	0	0	894	375		P-53204	FAngelix	0	0	158	784	
L-764db	Angelix	10	0	377	1	×1.09	P-5bb0a	Angelix	0	0	5 106	6162	×6.55
L-764db	FAngelix	10	0	348	3		P-5bb0a	FAngelix	0	0	779	1172	
L-827b6	Angelix	0	0	1 602	100	×6.48	P-63673	Angelix	10	4	451	56	×2.45
L-827b6	FAngelix	0	0	247	122		P-63673	FAngelix	10	10	184	1	
L-96a5f	Angelix	10	0	405	5	×1.12	P-70075	Angelix	10	0	434	40	×2.1
L-96a5f	FAngelix	10	0	362	1		P-70075	FAngelix	10	0	206	3	
L-a72cf	Angelix	10	0	3 600	217	×19.22	P-793cf	Angelix	0	0	14 400	0	×18.9
L-a72cf	FAngelix	10	0	187	4		P-793cf	FAngelix	0	0	762	117	
L-b2ce5	Angelix	0	0	548	273	×2.39	P-8138f	Angelix	10	5	415	7	×2.17
L-b2ce5	FAngelix	0	0	229	54		P-8138f	FAngelix	10	5	191	2	
L-ce4b7	Angelix	0	0	3 068	2735	×4.03	P-86efc	Angelix	0	0	1 161	161	×2.56
L-ce4b7	FAngelix	0	0	762	1290		P-86efc	FAngelix	0	0	453	582	
L-d59e7	Angelix	0	0	1 058	537	×1.96	P-a6c0a	Angelix	0	0	619	543	×1.61
L-d59e7	FAngelix	0	0	539	499		P-a6c0a	FAngelix	0	0	385	526	
L-e8a47	Angelix	10	10	322	35	×1.16	P-b60f5	Angelix	10	0	5 047	432	×23.11
L-e8a47	FAngelix	10	10	277	7		P-b60f5	FAngelix	10	0	218	9	
L-eb326	Angelix	10	0	482	7	×0.92	P-d890e	Angelix	10	0	2 758	71	×13.35
L-eb326	FAngelix	10	0	522	35		P-d890e	FAngelix	10	0	207	7	
L-ec4c	Angelix	10	0	3 624	186	×19.17	P-e65d3	Angelix	10	10	1 112	7	×4.95
L-ec4c	FAngelix	10	0	189	2		P-e65d3	FAngelix	10	10	224	15	
M-13421	Angelix	0	0	4 890	22	×4.82	P-eb0dd	Angelix	0	0	777	57	×1.9
M-13421	FAngelix	10	10	1 015	8		P-eb0dd	FAngelix	0	0	409	544	
M-14167	Angelix	10	5	887	24	×1.09	P-f912a	Angelix	0	0	516	17	×3.28
M-14167	FAngelix	10	4	816	21		P-f912a	FAngelix	0	0	158	10	
W-37111	Angelix	10	0	1 223	23	×1.3	P-fefe9	Angelix	0	0	2 624	141	×2.41
W-37111	FAngelix	10	0	940	64		P-fefe9	FAngelix	10	0	1 087	1812	
W-37123	Angelix	0	0	8 241	160	×3.87	Z-1a085	Angelix	0	0	1 806	763	×3.95
W-37123	FAngelix	0	0	2 130	3870		Z-1a085	FAngelix	0	0	457	120	
W-37171	Angelix	0	0	7 624	29	×15.18	Z-3cb60	Angelix	10	0	478	211	×1.96
W-37171	FAngelix	10	0	502	24		Z-3cb60	FAngelix	10	10	244	155	
W-37173	Angelix	10	0	6 745	46	×14.4	Z-3fe0c	Angelix	0	0	8 108	31	×10.65
W-37173	FAngelix	10	0	468	4		Z-3fe0c	FAngelix	10	0	761	564	
W-37285	Angelix	10	0	1 811	284	×3.59	Z-a1d3d	Angelix	10	0	507	340	×3.65
W-37285	FAngelix	10	0	504	37		Z-a1d3d	FAngelix	10	0	139	11	

[0124]

[0126] 도 4는 제약 조건 기반의 자동 프로그램 수정을 수행하는 분석장치(100)에 대한 예이다. 분석장치(100)는 물리적으로 다양한 형태로 구현될 수 있다. 예컨대, 분석장치(100)는 PC와 같은 컴퓨터 장치, 네트워크의 서버, 데이터 처리 전용 칩셋, 스마트기기 등의 형태로 구현될 수 있다.

[0127] 분석장치(100)는 저장장치(110), 메모리(120), 연산장치(130), 인터페이스 장치(140), 통신장치(150) 및 출력장치(160)를 포함할 수 있다.

[0128] 저장장치(110)는 수정 대상인 버그가 있는 프로그램 소스 파일(소스 코드)을 저장할 수 있다.

[0129] 저장장치(110)는 표현식 테스트를 위한 테스트 스위트(test suite)를 저장할 수 있다.

- [0130] 저장장치(110)는 자동 프로그램 수정 과정을 제어하는 코드 내지 프로그램을 저장할 수 있다. 저장장치(110)는 소스 코드에서 의심스러운 표현식(후보 표현식)을 식별하기 위한 SBFL 프로그램 내지 코드 저장할 수 있다. 저장장치(110)는 명세 추론을 위한 프로그램 내지 코드를 저장할 수 있다. 저장장치(110)는 명세 추론 과정에서 전술한 후보 경로를 샘플링하기 위한 프로그램 내지 코드 저장할 수 있다. 또한, 저장장치(110)는 추론된 명세에 따라 패치를 생성하는 프로그램 내지 코드를 저장할 수 있다.
- [0131] 저장장치(110)는 버그가 수정된 소스 코드를 저장할 수 있다.
- [0132] 메모리(120)는 분석장치가 소스 코드의 버그를 수정하는 과정에서 생성되는 데이터 및 정보 등을 저장할 수 있다.
- [0133] 인터페이스 장치(140)는 외부로부터 일정한 명령 및 데이터를 입력받는 장치이다. 인터페이스 장치(140)는 물리적으로 연결된 입력 장치 또는 외부 저장장치로부터 버그가 있는 프로그램의 소스 코드를 입력받을 수 있다. 인터페이스 장치(140)는 버그가 수정된 프로그램의 소스 코드를 외부 객체에 전달할 수도 있다.
- [0134] 통신장치(150)는 유선 또는 무선 네트워크를 통해 일정한 정보를 수신하고 전송하는 구성을 의미한다. 통신장치(150)는 외부 객체로부터 버그가 있는 프로그램의 소스 코드를 수신할 수 있다. 또는 통신장치(150)는 버그가 수정된 프로그램의 소스 코드를 외부 객체에 송신할 수도 있다.
- [0135] 인터페이스 장치(140) 및 통신장치(150)는 사용자 또는 다른 물리적 객체로부터 일정한 데이터를 주고받는 구성이므로, 포괄적으로 입출력장치라고도 명명할 수 있다. 정보 내지 데이터 입력 기능에 한정하면 인터페이스 장치(140) 및 통신장치(150)는 입력장치라고 할 수도 있다.
- [0136] 출력장치(160)는 일정한 정보를 출력하는 장치이다. 출력장치(160)는 코드 내용, 오류 수정 과정에 필요한 인터페이스, 버그가 수정된 코드 등을 출력할 수 있다.
- [0137] 연산 장치(130)는 SBFL을 이용하여 소스 코드에서 후보 표현식들을 식별할 수 있다. 즉, 연산 장치(130)는 소스 코드에서 결합 위치를 식별할 수 있다.
- [0138] 연산 장치(130)는 패치 생성을 위한 명세를 추론할 수 있다. 연산 장치(130)는 후보 표현식들에 대하여 테스트를 통과하는 후보 경로를 추출할 수 있다. 테스트는 사전에 마련된 테스트 스위트에서 선별한 특정 테스트 케이스들 중 적어도 하나일 수 있다.
- [0139] 후보 경로 탐색은 전술한 설명과 같다. 연산 장치(130)는 MCMC 샘플링 기법을 이용하여 후보 경로를 탐색할 수 있다.
- [0140] 연산 장치(130)는 MCMC 샘플링을 사용하여 심볼릭 실행을 사용하지 않을 수 있다. 이를 위하여 연산 장치(130)는 존재하는 프로그램에서 의심스러운 각 표현식을 래퍼 함수를 사용하여 변환할 수 있다.
- [0141] 연산 장치(130)는 후보 경로를 나타내는 비트 벡터 시퀀스를 무작위로 변환하면서 비용을 기준으로 설정을 저장한다. 전술한 바와 같이 설정에 포함되는 후보 경로의 벡터 길이가 다를 수 있다. 연산 장치(130)는 테스트 과정에서 후보 표현식 e 가 실행되는 횟수가 현재 설정 S '의 명세된 값들보다 많은 경우(명세된 값들이 모두 소모된 경우), 해당 후보 표현식이 실행되면 임의의 비트값을 리턴한다. 반대로 후보 표현식 e 의 실행 횟수가 $|S'|$ 보다 작다면 연산 장치(130)는 S' 에서 실제 사용된 비트값들을 추출하여(즉 사용된 일부를 선택하여) 새로운 S^* 로 설정을 조정할 수 있다.
- [0142] 이 과정에서 연산 장치(130)는 후보 경로에 대한 비용을 기준으로 후보 경로를 탐색하는 과정을 진행할 수 있다. 연산 장치(130)는 현재 후보 경로에 대한 비용 C 와 새로운 후보 경로에 대한 비용 C^* 을 비교하여 $C^* < C$ 라면, S^* 를 새로운 설정으로 선택한다. 또한 $C^* > C$ 인 경우에도 p 의 확률로 S^* 를 새로운 설정으로 선택한다. p 는 $\alpha(S \rightarrow S^*)$ 로 계산되며 $\alpha(S \rightarrow S^*)$ 는 수학식 3을 통해 계산된다. 비용 연산은 전술한 수학식 3에서 설명한 바와 같다. 비용 함수는 사용자가 설정한 특정 함수를 이용할 수 있다.
- [0143] 연산 장치(130)는 다수의 후보 경로를 탐색하지 않고, 후보 경로의 실제 출력값과 기대하는 출력값이 같은 경우 더 이상 후보 경로를 찾지않고 탐색을 종료할 수 있다.
- [0144] 연산 장치(130)는 탐색한 후보 경로에 대한 후처리 과정을 수행할 수 있다. 연산 장치(130)는 현재 후보 경로를 발견한 후보 설정 S 를 초기 프로그램의 실행 경로에서 획득가능한 초기 설정 S_1 과 비교하여 주어진 테스트를 통

과하면서 가장 최소의 비트 변화만을 갖도록 설정을 조정할 수 있다. 즉, 연산 장치(130)는 테스트 통과를 조건으로 초기 설정과 가장 유사하도록 설정을 조정할 수 있다. 후처리된 설정은 하나의 비트가 변경되어도 후보 경로가 되지 못하는 설정(1-minimal configuration)이 된다.

[0145] 명세 추론 과정이 종료되면, 연산 장치(130)는 패치 합성 프로그램 내지 코드를 이용하여 소스 코드 수정을 위한 패치를 생성할 수 있다. 이후 연산 장치(130)는 패치를 이용하여 소스 코드의 버그를 수정할 수 있다.

[0146] 연산 장치(130)는 데이터를 처리하고, 일정한 연산을 처리하는 프로세서, AP, 프로그램이 임베디드된 칩과 같은 장치일 수 있다.

[0148] 또한, 상술한 바와 같은 자동 프로그램 수정 내지 명세 추론 방법은 컴퓨터에서 실행될 수 있는 실행가능한 알고리즘을 포함하는 프로그램(또는 어플리케이션)으로 구현될 수 있다. 상기 프로그램은 일시적 또는 비일시적 판독 가능 매체(non-transitory computer readable medium)에 저장되어 제공될 수 있다.

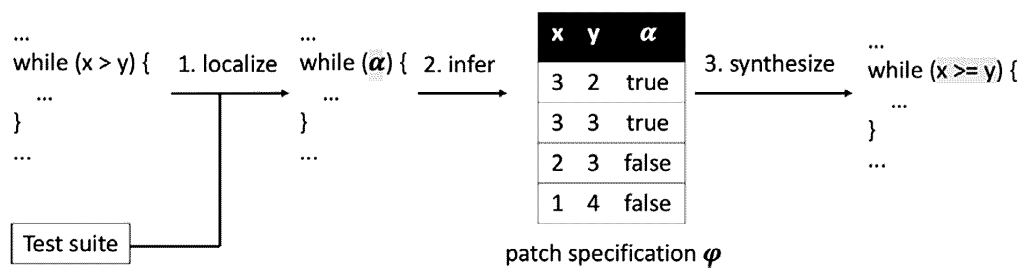
[0149] 비일시적 판독 가능 매체란 레지스터, 캐쉬, 메모리 등과 같이 짧은 순간 동안 데이터를 저장하는 매체가 아니라 반영구적으로 데이터를 저장하며, 기기에 의해 판독(reading)이 가능한 매체를 의미한다. 구체적으로는, 상술한 다양한 어플리케이션 또는 프로그램들은 CD, DVD, 하드 디스크, 블루레이 디스크, USB, 메모리카드, ROM (read-only memory), PROM (programmable read only memory), EPROM(Erasable PROM, EPROM) 또는 EEPROM(Electrically EPROM) 또는 플래시 메모리 등과 같은 비일시적 판독 가능 매체에 저장되어 제공될 수 있다.

[0150] 일시적 판독 가능 매체는 스태틱 램(Static RAM, SRAM), 다이내믹 램(Dynamic RAM, DRAM), 싱크로너스 디램(Synchronous DRAM, SDRAM), 2배속 SDRAM(Double Data Rate SDRAM, DDR SDRAM), 증강형 SDRAM(Enhanced SDRAM, ESDRAM), 동기화 DRAM(Synclink DRAM, SLDRAM) 및 직접 램버스 램(Direct Rambus RAM, DRRAM) 과 같은 다양한 RAM을 의미한다.

[0151] 본 실시예 및 본 명세서에 첨부된 도면은 전술한 기술에 포함되는 기술적 사상의 일부를 명확하게 나타내고 있는 것에 불과하며, 전술한 기술의 명세서 및 도면에 포함된 기술적 사상의 범위 내에서 당업자가 용이하게 유추할 수 있는 변형 예와 구체적인 실시례는 모두 전술한 기술의 권리범위에 포함되는 것이 자명하다고 할 것이다.

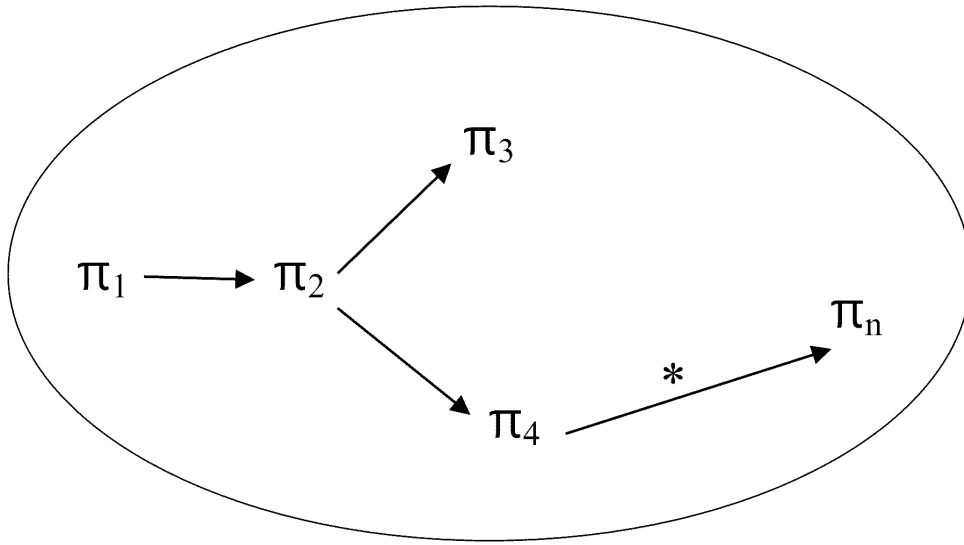
도면

도면1

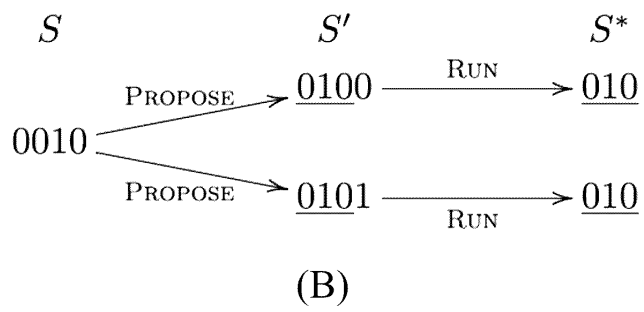
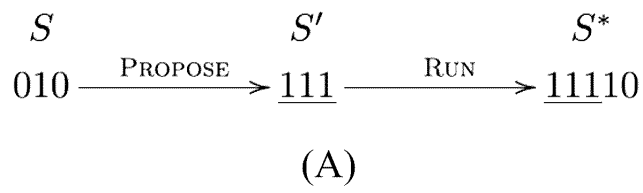


도면2

[실행 경로 공간]



도면3



도면4

