



(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2013년09월10일
(11) 등록번호 10-1306842
(24) 등록일자 2013년09월04일

(51) 국제특허분류(Int. Cl.)

G06F 9/44 (2006.01) G06F 9/45 (2006.01)

(21) 출원번호 10-2012-0074846

(22) 출원일자 2012년07월10일

심사청구일자 2012년07월10일

(56) 선행기술조사문헌

자바 프로그램 검증을 위한 스택 리스 중간 표현 언어 생성기 구현, 김선태 외 3명, 한국정보기술 학회논문지 제9권 제9호, 2011년 09월 JP07192034 A

(73) 특허권자

인하대학교 산학협력단

인천광역시 남구 인하로 100, 인하대학교 (용현동)

(72) 발명자

유원희

서울특별시 강서구 화곡3동 화곡푸르지오@113-802

박준석

서울특별시 서초구 서초1동 1642-34 래미안 서초 7차 아파트 102동 602호

(뒷면에 계속)

(74) 대리인

특허법인엠에이피에스

전체 청구항 수 : 총 13 항

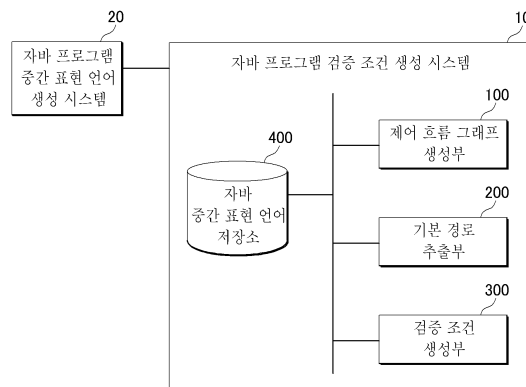
심사관 : 최정권

(54) 발명의 명칭 자바 프로그램 검증 조건 생성 시스템 및 방법

(57) 요약

자바 프로그램 중간 표현 언어를 저장하는 자바 중간 표현 언어 저장소; 상기 자바 중간 표현 언어로부터 제어 흐름 그래프를 생성하는 제어 흐름 그래프 생성부; 상기 제어 흐름 그래프로부터 기본 경로를 추출하는 기본 경로 추출부; 및 상기 기본 경로로부터 검증 조건을 생성하는 검증 조건 생성부;를 포함하는 자바 프로그램 검증 조건 생성 시스템을 제공한다.

대표도 - 도1



(72) 발명자

김제민

인천광역시 부평구 십정동 181-112 2층

김선태

인천광역시 남구 용현동 31-21번지 1층

특허청구의 범위

청구항 1

자바 프로그램 검증 조건 생성 시스템에 있어서,
 자바 프로그램 중간 표현 언어를 저장하는 자바 중간 표현 언어 저장소;
 상기 자바 중간 표현 언어로부터 제어 흐름 그래프를 생성하는 제어 흐름 그래프 생성부;
 상기 제어 흐름 그래프로부터 기본 경로를 추출하는 기본 경로 추출부; 및
 상기 기본 경로로부터 검증 조건을 생성하는 검증 조건 생성부;를 포함하는 자바 프로그램 검증 조건 생성 시스템.

청구항 2

제 1 항에 있어서,
 상기 자바 프로그램 중간 표현 언어는
 프로그램 검증을 위한 전조건, 후조건, 및 루프불변자를 포함하는 명세 정보를 입력한 자바 프로그램 소스 파일을 변환하여 생성되는 것이며
 상기 명세 정보 입력에는 JML(Java Modeling Language) 문법이 사용되는 자바 프로그램 검증 조건 생성 시스템.

청구항 3

제 1 항에 있어서,
 상기 자바 프로그램 중간 표현 언어는
 자바 프로그램 바이트코드 파일을 변환하여 생성되는 것인 자바 프로그램 검증 조건 생성 시스템.

청구항 4

제 1 항에 있어서,
 상기 제어 흐름 그래프는 상기 자바 프로그램 중간 표현 언어의 각 블록을 노드(node)로 하고, 분기를 간선(edge)으로 하는 순서가 있는 그래프이며,
 상기 간선에는 상기 분기시 만족해야 하는 조건이 포함되는 자바 프로그램 검증 조건 생성 시스템.

청구항 5

제 2 항에 있어서,
 상기 제어 흐름 그래프의 시작 노드는 상기 전조건을 나타내고, 종료 노드는 상기 후조건을 나타내도록 구성되는 자바 프로그램 검증 조건 생성 시스템.

청구항 6

제 1 항에 있어서,
 상기 기본 경로는 상기 제어 흐름 그래프의 시작 노드 또는 루프불변자 블록을 나타내는 노드로부터 시작되고,
 상기 제어 흐름 그래프의 종료 노드, 단언(assert) 블록을 나타내는 노드 또는 루프불변자 블록을 나타내는 노드에서 끝나며,
 상기 제어 흐름 그래프를 깊이 우선 탐색하여 추출되는 자바 프로그램 검증 조건 생성 시스템.

청구항 7

제 1 항에 있어서,

상기 제어 흐름 그래프의 간선에 포함되어 있는 조건은 상기 기본 경로에서 가정(assume) 표현식으로 변환되는 자바 프로그램 검증 조건 생성 시스템.

청구항 8

제 1 항에 있어서,

상기 검증 조건 생성부는 최약 전조건(weakest precondition) 계산 함수를 사용하여 상기 검증 조건을 생성하며,

상기 최약 전조건 계산 함수는 상기 기본 경로에 포함되어 있는 각 표현식을 인수로 하여 순환 호출되는 자바 프로그램 검증 조건 생성 시스템.

청구항 9

제 8 항에 있어서,

상기 최약 전조건 계산 함수는 상기 순환 호출시 상기 기본 경로의 끝에 있는 표현식을 초기 후조건 인수로 받는 자바 프로그램 검증 조건 생성 시스템.

청구항 10

자바 프로그램 검증 조건 생성 시스템을 이용한 자바 프로그램 검증 조건 생성 방법에 있어서,

자바 중간 표현 언어로부터 제어 흐름 그래프를 생성하는 단계;

상기 제어 흐름 그래프로부터 기본 경로를 추출하는 단계; 및

상기 기본 경로로부터 검증 조건을 생성하는 단계;를 포함하는 자바 프로그램 검증 조건 생성 방법.

청구항 11

제 10 항에 있어서,

상기 제어 흐름 그래프는 상기 자바 프로그램 중간 표현 언어의 각 블록을 노드(node)로 하고, 분기를 간선(edge)으로 하는 순서가 있는 그래프이며,

상기 간선에는 상기 분기시 만족해야 하는 조건이 포함되는 자바 프로그램 검증 조건 생성 방법.

청구항 12

제 10 항에 있어서,

상기 기본 경로는 상기 제어 흐름 그래프의 시작 노드 또는 루프불변자 블록을 나타내는 노드로부터 시작되고,

상기 제어 흐름 그래프의 종료 노드, 단언(assert) 블록을 나타내는 노드 또는 루프불변자 블록을 나타내는 노드에서 끝나며,

상기 제어 흐름 그래프를 깊이 우선 탐색하여 추출되는 자바 프로그램 검증 조건 생성 방법.

청구항 13

제 10 항에 있어서,

상기 검증 조건 생성에는 최약 전조건(weakest precondition) 계산 함수가 사용되며,

상기 최약 전조건 계산 함수는 상기 기본 경로에 포함되어 있는 각 표현식을 인수로 하여 순환 호출되는 자바 프로그램 검증 조건 생성 방법.

명세서

기술분야

[0001] 본 발명은 자바 프로그램 검증 조건 생성 시스템 및 방법에 관한 것이다.

배경 기술

[0002] 자바 언어는 효율적인 개발환경, 재사용성, 안전한 실행, 플랫폼 독립성 등 많은 장점을 가지고 있어, C 언어, C++ 언어 등과 더불어 현재 가장 많이 이용되는 언어가 되었다. 특히 자바는 항공, 우주, 핵 발전소 등과 같은 최신 분야에서 널리 사용한다. 이와 같은 환경에서 수행되는 프로그램은 안전하고 오류 없이 실행되는 것이 중요하다. 그러므로 프로그램 수행 전에 오류 없이 안전하게 실행하는 것이 보장되어야 한다.

[0003] 따라서 자바 프로그램을 수행하기 전에 프로그램의 안전한 실행을 보장하기 위한 테스트 및 검증 방법이 필요하며, 이에 대한 연구가 활발히 진행되어 왔다.

[0004] 이와 관련하여 한국공개특허 제10-2004-0050463호("엔터프라이즈 자바 빈 컴포넌트 테스트 장치 및 그 방법")은 객체 지향언어로 구현된 서버 컴포넌트를 테스트 하고자 하는 경우에 다양한 테스트 케이스와 데이터를 자동 생성해 주고, 이를 XML문서로 생성하는 구성을 개시하고 있다.

[0005] 또한, 미국공개특허 제2009-0037884호("Method for Automatically Creating a Behavior Pattern of a Computer Program for Model-based Testing Techniques")은 주식과 함께 제공되는 소스 코드를 포함하는 컴퓨터 프로그램에 대해 자동으로 동작(behavior) 패턴을 생성해주는 구성을 개시하고 있다.

발명의 내용

해결하려는 과제

[0006] 본 발명은 전술한 자바 프로그램 검증 문제를 해결하기 위한 것으로서, 그 목적은 자바 프로그램 중간 표현 언어를 검증 조건으로 변환하는 자바 프로그램 검증 조건 생성 시스템 및 방법을 제공하는 것이다.

과제의 해결 수단

[0007] 상기와 같은 목적을 달성하기 위한 본 발명의 제 1 측면에 따른 자바 프로그램 검증 조건 생성 시스템은 자바 프로그램 중간 표현 언어를 저장하는 자바 중간 표현 언어 저장소; 상기 자바 중간 표현 언어로부터 제어 흐름 그래프를 생성하는 제어 흐름 그래프 생성부; 상기 제어 흐름 그래프로부터 기본 경로를 추출하는 기본 경로 추출부; 및 상기 기본 경로로부터 검증 조건을 생성하는 검증 조건 생성부;를 포함하는 것을 특징으로 한다.

[0008] 상기와 같은 목적을 달성하기 위한 본 발명의 제 2 측면에 따른 자바 프로그램 검증 조건 생성 시스템을 이용한 자바 프로그램 검증 조건 생성 방법은 자바 중간 표현 언어로부터 제어 흐름 그래프를 생성하는 단계; 상기 제어 흐름 그래프로부터 기본 경로를 추출하는 단계; 및 상기 기본 경로로부터 검증 조건을 생성하는 단계;를 포함하는 것을 특징으로 한다.

발명의 효과

[0009] 본 발명에 따른 자바 프로그램 검증 조건 생성 시스템 및 자바 프로그램 검증 방법은 자바 프로그램에 대한 프로그램 유효성을 검증할 수 있는 검증 조건을 생성할 수 있다.

[0010] 구체적으로, 프로그램 검증을 위한 명세를 포함하는 자바 프로그램에 대한 자바 프로그램 중간 표현 언어를 검증 조건으로 간단하게 변환할 수 있다. 따라서 프로그래머는 자바 프로그램이 실행 중 만족해야 할 조건을 명세하여 자동 검증할 수 있다.

도면의 간단한 설명

[0011] 도 1은 본 발명의 일실시예에 따른 자바 프로그램 검증 조건 생성 시스템의 구조를 도시함.

도 2는 기본 경로 추출 알고리즘의 일례를 도시함.

도 3은 최약 전조건 계산 함수의 명세를 도시함.

도 4에서 도 8는 본 발명에 따른 실시예를 도시함.

도 9는 본 발명의 일실시예에 따른 자바 프로그램 검증 조건 생성 방법의 흐름을 도시함.

발명을 실시하기 위한 구체적인 내용

- [0012] 아래에서는 첨부한 도면을 참조하여 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자가 용이하게 실시할 수 있도록 본 발명의 실시 예를 상세히 설명한다. 그러나 본 발명은 여러 가지 상이한 형태로 구현될 수 있으며 여기에서 설명하는 실시 예에 한정되지 않는다. 그리고 도면에서 본 발명을 명확하게 설명하기 위해서 설명과 관계없는 부분은 생략하였으며, 명세서 전체를 통하여 유사한 부분에 대해서는 유사한 도면 부호를 붙였다.
- [0013] 명세서 전체에서, 어떤 부분이 다른 부분과 "연결"되어 있다고 할 때, 이는 "직접적으로 연결"되어 있는 경우뿐 아니라, 그 중간에 다른 소자를 사이에 두고 "전기적으로 연결"되어 있는 경우도 포함한다. 또한 어떤 부분이 어떤 구성요소를 "포함"한다고 할 때, 이는 특별히 반대되는 기재가 없는 한 다른 구성요소를 제외하는 것이 아니라 다른 구성요소를 더 포함할 수 있는 것을 의미한다.
- [0014] 도 1은 본 발명의 일실시예에 따른 자바 프로그램 검증 조건 생성 시스템(10)의 구조를 도시하고 있다.
- [0015] 자바 프로그램 검증 조건 생성 시스템(10)은 제어 흐름 그래프 생성부(100), 기본 경로 추출부(200), 검증 조건 생성부(300), 자바 중간 표현 언어 저장소(400)를 포함한다.
- [0016] 우선, 자바 중간 표현 언어 저장소(400)에는 자바 프로그램 중간 표현 언어 생성 시스템(20)으로부터 수신한 자바 중간 표현 언어가 저장된다. 여기에서 자바 프로그램 중간 표현 언어는 BIRS(Bytecode Intermediate Representation Specification)이다.
- [0017] 이 자바 프로그램 중간 표현 언어는 자바 프로그램 검증 조건을 자바 소스 파일을 활용한 고급 수준에서는 물론 자바 클래스 파일(바이트코드: bytecode)을 활용한 저급 수준에서도 가능하게 하기 위해 본 발명의 발명자에 의해 개발된 언어이다. 즉, 자바 프로그램 검증을 위한 명세를 자바 소스 파일에 포함시키는 것이 프로그래머에게 더 편리한 방법이지만, 많은 경우 일부 또는 전체 모듈의 자바 소스 파일 없이 자바 바이트코드만을 가지고 검증해야 할 필요가 있기 때문에, 본 발명의 발명자는 유연함을 위하여 이를 위한 자바 프로그램 중간 표현 언어 및 자바 프로그램 중간 표현 언어 생성 시스템(20)을 발명하였다.
- [0018] 자바 소스 없이 자바 바이트코드만을 가지고 프로그램을 개발하는 경우, 역 컴파일을 통해 자바 바이트코드로부터 자바 소스를 얻는다고 해도 컴파일 과정 중 손실되는 정보들을 완전히 복구할 수는 없기 때문에, 자바 바이트코드만으로 해당 프로그램을 검증해야 할 때 자바 프로그램 중간 표현 언어 생성 시스템(20)을 사용하여 자바 프로그램 중간 표현 언어를 생성하면 프로그램을 검증하기에 편리하다. 명세정보가 입력되지 않은 자바 프로그램 소스가 자바 프로그램 중간 표현 언어로 변환되었을 경우 사용자가 직접 명세를 작성하는 것이 가능하며, 명세 정보 입력에는 BIRS 언어 명세 문법이 사용된다.
- [0019] 자바 소스가 있을 때는 자바 소스 파일에 전조건(pre-condition), 후조건(post-condition), 루프불변자(loop invariant) 등의 명세 정보를 기술할 수 있다. 전조건은 프로그램의 메소드가 실행하기 전에 만족해야 하는 성질을, 후조건은 프로그램의 메소드가 실행후 만족해야 하는 성질을 나타낸다. 루프불변자는 메소드 내에 존재하는 While 문과 같은 루프문에 대해 루프문의 반복마다 만족해야 하는 성질을 나타내기 위해 사용된다.
- [0020] 이러한 명세 정보는 명세 언어를 통해 기술된다. 예컨대, JML(Java Modeling Language)이 프로그램 검증 조건을 기술하는 언어로 사용할 수 있다(JML2BML 도구를 사용하여 BML이 생성되고, 생성된 BML을 입력으로 받아 BIRS를 생성함). JML은 자바 소스 코드에 프로그래머가 전조건과 후조건, 루프불변자 등을 주석 형태로 명세가 가능한 언어이다. 즉, 자바 소스 파일에서 @로 시작하는 주석(/**@ 또는 /*@)은 JML 문법을 사용하여 기술된 명세 정보이다. 주석 형태로 명세된 정보는 프로그램이 수행되기 전, 메소드의 의도된 기능을 설명한다.
- [0021] 본 발명에서 이러한 전조건, 후조건, 및 루프불변자 등의 프로그램 검증을 위한 명세는 플로이드-호어 방법(Floyd-Hoare Logic 또는 Hoare Rule)을 따른다. 플로이드 호어 방법은 명세와 참거짓 자동 판별기를 이용하는 방법이다. 프로그래머는 프로그램 소스 코드에 대해 해당 프로그램 실행시 만족해야 할 조건을 명세한 후, 해당 명세를 검증 조건으로의 변환, 변환된 검증 조건의 참거짓 자동 판별기를 사용한 유효성 검사를 통해 해당 프로그램이 자신의 의도에 맞게 동작할지 여부를 검증할 수 있다. 본 발명에 따른 자바 프로그램 검증 조건 생성 시스템(10)은 이러한 유효성 검사에 사용될 검증 조건을 생성한다.
- [0022] 먼저, 제어 흐름 그래프 생성부(100)는 자바 중간 표현 언어 저장소(400)에 저장되어 있는 자바 중간 표현 언어로부터 제어 흐름 그래프를 생성한다.
- [0023] 제어 흐름 그래프는 자바 프로그램 중간 표현 언어에서 사용되는 각 블록을 노드(node)로 하고, 분기를 간선

(edge)으로 하는 순서가 있는 그래프이며, 그래프의 간선에는 분기시 만족해야 하는 조건이 포함된다. 제어 흐름 그래프의 시작 노드(init)는 해당 모듈이 만족해야 할 전조건을 나타내고, 종료 노드(exit)는 후조건을 나타내도록 구성된다. 이때 전조건은 JML의 require 문, 후조건은 ensure 문이다.

[0024] 기본 경로 추출부(200)는 제어 흐름 그래프로부터 기본 경로를 추출한다.

[0025] 기본 경로는 제어 흐름 그래프의 시작 노드 또는 루프불변자 블록을 나타내는 노드로부터 시작되고, 제어 흐름 그래프의 종료 노드, 단언(assert) 블록을 나타내는 노드 또는 루프불변자 블록을 나타내는 노드에서 끝나며, 제어 흐름 그래프를 깊이 우선 탐색하여 추출된다. 이 과정에서 제어 흐름 그래프의 간선에 포함되어 있는 조건은 기본 경로에서는 가정(assume) 문으로 변환된다.

[0026] 도 2는 기본 경로 추출 알고리즘의 일례를 도시하고 있다.

[0027] 도면을 참조하면, 알고리즘은 노드집합(B), 간선집합(E), 시작노드(B_{init}), 종료노드(B_{exit})를 포함하는 그래프(G)를 입력받고, 하나 이상의 기본 경로(basic paths)를 출력하고 있다. traverse() 함수가 깊이 우선 탐색을 통하여 기본 경로를 추출한다. 이 함수는 그래프(G)의 노드집합(B)과 경로(P)를 인수로 받으며, 경로(P)가 완성될 때까지 순환 호출된다. 순환 호출시의 초기 인수는 그래프(G)의 시작노드(B_{init})이다.

[0028] 다시 도 1로 돌아가서, 검증 조건 생성부(300)는 기본 경로로부터 검증 조건을 생성한다.

[0029] 검증 조건 생성에는 최약 전조건(weakest precondition) 계산 함수가 사용되며, 최약 전조건 계산 함수는 기본 경로에 포함되어 있는 각 표현식을 인수로 하여 순환 호출된다. 이때 최약 전조건 계산 함수는 순환 호출시 기본 경로의 끝에 있는 표현식을 초기 후조건 인수로 받는다.

[0030] 도 3은 최약 전조건 계산 함수의 명세를 도시하고 있다.

[0031] 최약 전조건 계산 함수 wp()는 도시되어 있는 바와 같이, 표현식 inst와 후조건 Q를 인수로 받는다. 전술한 바와 같이, 초기 후조건 인수는 기본 경로의 끝에 있는 표현식이며, 기본 경로는 그래프(G)의 종료노드(B_{exit}), 즉, 후조건(ensure) 문, 또는 루프불변자(loopInvariant) 블록이나 단언(assert) 블록을 나타내는 노드로 끝난다.

[0032] 도시되어 있는 바와 같이, assume Expr 표현식은 후조건 Q에 대해 Expr이 참일 경우에만 Q가 참인지 거짓인지 중요하므로 검증 조건으로 변환하면 $Expr \Rightarrow Q$ 와 같다. assert Expr 표현식은 Expr이 참이고 후조건 Q도 참이어야 하므로 $Expr \ \&\& \ Q$ 로 변환된다. 배정문 $VarId := Expr$ 의 경우 후조건 Q의 VarId를 Expr로 치환함으로써 최약 전조건을 구한다.

[0033] 이런 식으로 표현식의 나열 $Ins_1; \dots ; Ins_n$ 의 경우 후조건 Q에 대해 최약 전조건 계산 함수 wp()를 순환 호출함으로써 각 기본 경로로부터 검증 조건을 도출한다.

[0034] 도 4내지 도 8은 본 발명에 따른 실시예를 도시하고 있다.

[0035] 도 4는 자바 소스 프로그램의 실시예를 도시하고 있다.

[0036] 실시예의 Search 클래스는 배열의 인덱스 l부터 u까지를 탐색 하여 원소 e를 찾는 linearSearch 메소드를 포함하고 있다. 전술한 바와 같이, 이 메소드가 프로그래머가 의도한대로 동작하는지 검증하기 위한 명세가 JML을 통해 명세되어 있다.

[0037] linearSearch 메소드가 호출되기 위해서는 l은 0보다 크거나 같아야 하고 u는 배열 a의 길이보다 작아야 한다. 이러한 성질을 나타내기 위해 3행에서 JML의 requires 절을 통해 함수가 호출되기 전에 만족해야 하는 전조건을 나타내고 있다.

[0038] linearSearch 메소드가 호출된 후의 결과값은 l과 u사이의 배열 a에서 원소 e를 찾았을 경우 참(true)이고 찾지 못했을 경우거짓(false)이다. 이러한 성질을 나타내기 위해 4행에서 JML의 ensures 절을 통해 함수가 호출되고 난 후 만족해야 하는 후조건을 나타내고 있다.

[0039] linearSearch 메소드의 for 문을 반복하는 동안 인덱스 변수 i는 l보다 크거나 같아야 하며 l과 i사이의 배열의 원소는 찾고자 하는 원소 e가 아님을 보장해야 한다. 이러한 성질을 나타내기 위해 6행에서 JML의 loop_variant 절을 통해 루프 불변자를 나타내는 명세를 표기하고 있다.

[0040] 도 5는 도 4의 자바 소스 프로그램을 변환한 자바 프로그램 중간 표현 언어의 실시예를 도시하고 있다.

- [0041] 생성된 자바 프로그램 중간 표현 언어(BIRS)는 for 문 등의 루프(loop) 대신 goto 문을 가지는 저급 언어이다. 도시된 바와 같이 루프 불변자 등의 명세가 번호가 붙은 블록으로 변환되었다.
- [0042] 도 6은 도 5의 자바 프로그램 중간 표현 언어에서 도출한 제어 흐름 그래프의 실시예를 도시하고 있다.
- [0043] 도시되어 있는 제어 흐름 그래프에서 각 노드의 번호는 도 5의 기본 블록의 번호를 나타낸다. 전술한 바와 같이, ensure 절과 requires 절을 나타내기 위해 시작 노드(init 블록)와 종료 노드(exit 블록)가 추가되었다.
- [0044] 블록 7과 블록 2에 분기문인 if문을 포함하고 있기 때문에 제어 흐름 그래프의 노드 7과 노드 2에서는 분기가 발생하며 각 간선에는 분기가 실행되기 위해 만족해야 하는 조건이 포함되었다. 노드 7, 노드 2, 노드 6에서 루프가 발생하는데, 이는 도 4의 소스 코드의 루프로부터 변환된 것이다.
- [0045] 도 7은 도 6의 제어 흐름 그래프에서 도출한 기본 경로의 실시 예를 도시하고 있다.
- [0046] 생성된 제어 흐름 그래프로부터 기본 경로를 추출하기 위해서는 기본 경로의 시작점과 끝점을 찾아야 한다. 전술한 바와 같이, 기본 경로의 시작점은 함수 전조건(requires 절)이나 루프 불변자일 수 있고 기본 경로의 끝점은 루프 불변자, 단언(assert), 또는 함수 후조건(ensures 절)이 될 수 있다.
- [0047] 따라서 본 실시 예에서 기본 경로의 시작점 후보에는 requires 절(그래프의 시작 노드)과 블록 7의 루프불변자가 포함되며, 기본 경로의 끝점 후보에는 ensures 절(그래프의 종료 노드)과 블록 7의 루프불변자가 포함된다. 기본 경로의 시작점과 끝점의 후보를 대상으로 제어 흐름 그래프의 init 노드부터 깊이 우선 탐색(Depth First Search)를 수행하여 기본 경로의 끝점을 만나면 기본 경로 하나를 반환하고 새로운 시작점의 후보로부터 끝점을 찾아서 기본 경로를 반환할 때까지 깊이 우선 탐색을 반복한다.
- [0048] 또한 전술한 바와 같이, 깊이 우선 탐색 과정에서 간선에 나타나 있는 조건에 대해서는 assume 문으로 변환함으로써 제어 흐름이 발생하는 조건을 나타낸다.
- [0049] 모든 기본 경로를 찾아내면 기본 경로를 추출하는 과정이 마무리된다. 그 결과 도 5의 코드로부터 추출한 기본 경로는 도 7과 같다.
- [0050] 도 8은 도 7의 기본 경로에서 도출한 검증 조건의 실시 예를 도시하고 있다.
- [0051] 전술한 바와 같이, 추출된 기본 경로로부터 검증 조건을 생성하는 과정은 최약 전조건 함수의 순환 호출을 통해 이루어진다. 기본 경로에서 마지막 문장을 초기 후조건 인수로 두고 마지막에서 두 번째 문장부터 후조건을 만족하는 전조건을 계산한다. 마지막 문장부터 첫 문장까지 순차적으로 최약전조건을 계산함으로써 하나의 기본 경로에 대해 하나의 검증 조건을 생성할 수 있다.
- [0052] 도 7의 기본 경로에 대해 최약전조건 계산방법을 적용하여 나온 결과는 도 8과 같다. 도 7와 도 8의 각 기본 경로와 검증 조건은 대응된다. 예를 들어, 도 7의 기본 경로 (a)는 PRE 문, 배정문, LOOPINV 문 순으로 나열되어 있는데, 도 8의 검증 조건 (a)와 대응된다.
- [0053] 도 7의 (a)에 대한 검증 조건의 생성 과정은 LOOPINV 문부터 역순으로 진행된다. 우선 마지막 문장인 LOOPINV에 대해 LOOPINV 문의 표현식 부분만이 검증 조건으로 고려된다. 그 다음 배정문 local5:=local2에 대해 전 과정에서 나온 표현식의 local5 변수를 local2 변수로 치환한다. 그 결과 LOOPINV 문의 표현식 부분 중 local5가 local2로 치환된 결과가 검증 조건으로 나온다. 이제 PRE 문에 대해 PRE 문의 표현식 부분과 전 과정에서 생성된 검증 조건을 암시(==>)로 연결하면 도 8의 검증 조건 (a)가 결과로 나타난다.
- [0054] 최종적으로 검증 조건은 술어논리식 형태로 나타난다. 술어논리식 형태로 나타난 검증 조건에 대해 참거짓 자동 판별기를 적용하면 생성된 기본 경로에 대한 오류여부를 파악할 수 있다. 즉, 각 기본 경로에 대해 나타나는 검증 조건을 모두 만족한다면 도 5의 자바 프로그램 중간 표현 언어 코드는 명세를 한 프로그래머의 의도에 맞게 실행된다는 것을 보장한다.
- [0055] 도 9는 본 발명의 일실시예에 따른 자바 프로그램 검증 조건 생성 방법의 흐름을 도시하고 있다.
- [0056] 먼저, 자바 프로그램 중간 표현 언어로부터 제어 흐름 그래프를 생성한다(S910). 전술한 바와 같이, 제어 흐름 그래프는 자바 프로그램 중간 표현 언어의 각 블록을 노드(node)로 하고, 분기를 간선(edge)으로 하는 순서가 있는 그래프이다.
- [0057] 다음, 제어 흐름 그래프로부터 기본 경로를 추출한다(S920). 전술한 바와 같이, 기본 경로는 제어 흐름 그래프의 시작 노드 또는 루프불변자 블록을 나타내는 노드로부터 시작되고, 제어 흐름 그래프의 종료 노드, 단언

(assert) 블록을 나타내는 노드 또는 루프불변자 블록을 나타내는 노드에서 끝난다.

[0058] 다음, 기본 경로로부터 최약 전조건 함수를 사용하여 검증 조건을 생성한다(S930). 전술한 바와 같이, 검증 조건 생성에는 최약 전조건(weakest precondition) 계산 함수가 사용되며, 최약 전조건 계산 함수는 기본 경로에 포함되어 있는 각 표현식을 인수로 하여 순환 호출된다.

[0059] 전술한 본 발명의 설명은 예시를 위한 것이며, 본 발명이 속하는 기술분야의 통상의 지식을 가진 자는 본 발명의 기술적 사상이나 필수적인 특징을 변경하지 않고서 다른 구체적인 형태로 쉽게 변형이 가능하다는 것을 이해할 수 있을 것이다. 그러므로 이상에서 기술한 실시 예들은 모든 면에서 예시적인 것이며 한정적이 아닌 것으로 이해해야만 한다. 예를 들어, 단일형으로 설명되어 있는 각 구성 요소는 분산되어 실시될 수도 있으며, 마찬가지로 분산된 것으로 설명되어 있는 구성 요소들도 결합된 형태로 실시될 수 있다.

[0060] 본 발명의 범위는 상기 상세한 설명보다는 후술하는 특허청구범위에 의하여 나타내어지며, 특허청구범위의 의미 및 범위 그리고 그 균등 개념으로부터 도출되는 모든 변경 또는 변형된 형태가 본 발명의 범위에 포함되는 것으로 해석되어야 한다.

부호의 설명

[0061] 10: 자바 프로그램 검증 조건 생성 시스템

100: 제어 흐름 그래프 생성부

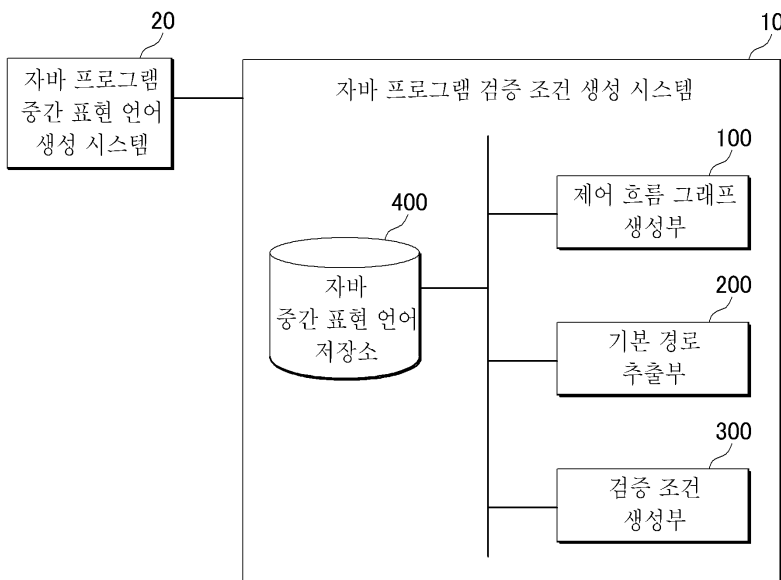
200: 기본 경로 추출부

300: 검증 조건 생성부

400: 자바 중간 표현 언어 저장소

도면

도면1



도면2

```

INPUT:  $G = B \times E \times \text{Binit} \times \text{Bexit}$ 
OUTPUT : Basic Paths
Method:  $\text{paths} := \emptyset$ 
         $\text{traverse}(\text{Binit}, \epsilon)$ 
        return paths
USING:
function  $\text{traverse}(\text{Node } N, \text{Path } P)$ 
{
    for  $i=1$  to  $N.\text{length}$  do
        if  $N[i]$  is loop invariant then
             $P := P ; N[i]$ 
             $\text{paths} := \text{paths} \cup \{P\}$ 
             $P := N[i]$ 
        else if  $N[i]$  is assert then
             $\text{tmp} := P$ 
             $P := P ; N[i]$ 
             $\text{paths} := \text{paths} \cup \{P\}$ 
             $P := \text{tmp}$ 
        else
             $P := P ; N[i]$ 
    if  $N$  is not marked then
         $N.\text{marked} := \text{true}$ 
         $\text{edges} := N.\text{succ}$ 
        for all  $e$  in  $\text{edges}$  do
            if  $e$  is conditional edge then
                 $P := P ; \text{assume } e.\text{cond}$ 
             $\text{Succ} := e.\text{head}$ 
             $\text{traverse}(\text{Succ}, P)$ 
}

```

도면3

inst	Q	$\text{wp}(\text{inst}, Q)$
assume Expr	Q	$\text{Expr} \implies Q$
assert Expr	Q	$\text{Expr} \ \&\& \ Q$
requires Expr	Q	$\text{Expr} \implies Q$
$\text{Ins}_1; \dots ; \text{Ins}_n$	Q	$\text{wp}(\text{Ins}_1; \dots \text{Ins}_{n-1}, \text{wp}(\text{Ins}_n, Q))$
$\text{VarId} := \text{Expr}$	$Q[\text{VarId}]$	$Q[\text{Expr}]$

도면4

```

1: package main;

2: public class Search{
3:   //@ requires 0 <= l && u < a.length;
4:   //@ ensures Wresult == (Wexists int j; l<=j && j<=u && a[j]==e);
5:   public boolean linearSearch(int[] a, int l, int u, int e){
6:     //@ loop_invariant l<=i && (Wforall int j; l<=j && j<i ==> a[j]!=e);
7:     for(int i=l; i<=u; i=i+1){
8:       if(a[i]==e)
9:         return true;
10:    }
11:    return false;
12:  }
13:}

```

도면5

```

Class main.Search: (java.lang.Object ) () ()
{computationalFunction linearSearch
( java.lang.Object local0, arrrint local1, int local2,
int local3, int local4 ) -> bool
require ((0 <= local2) && (local3 < local1.length))
ensure (result == (Exist int j. (((local2 <= j) &&
(j <= local3)) && (local1[j] == local4))))

block 8 is
8: vreturn 0
end

block 7 is
7: loopInvariant (((local2 <= local5) &&
(ForAll int j. (((local2 <= j) &&
(j < local5)) ==> not((local1[j] == local4))))))
7: ifd(local5 <= local3) 2
end

block 6 is
6: assignVal local5 := (local5 + 1)
end

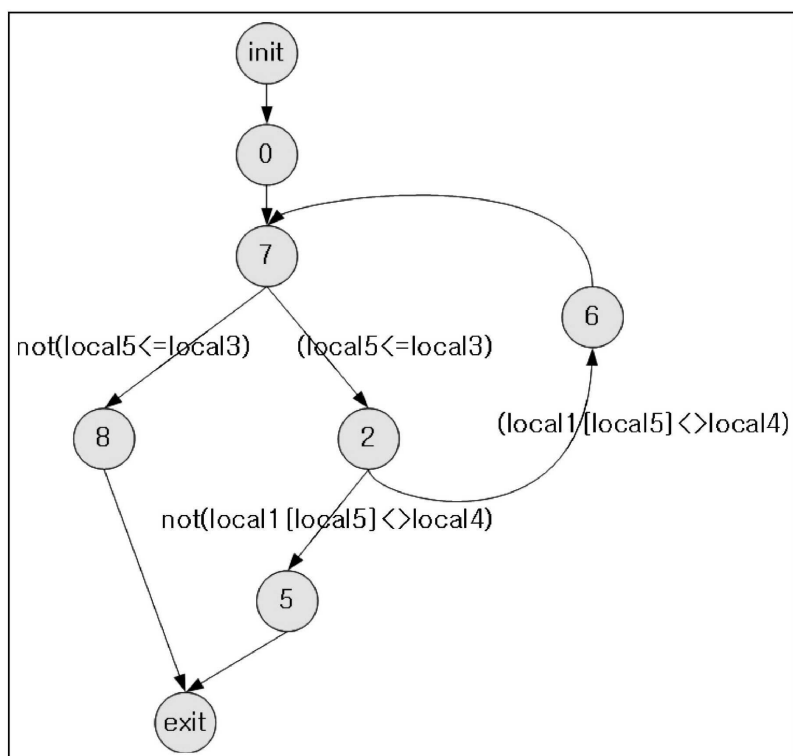
block 5 is
5: vreturn 1
end

block 2 is
2: ifd(local1[local5] <> local4) 6
end

block 0 is
0: assignVal local5 := local2
1: goto 7
end
block 7 ==> block 8, not local5 <= local3
block 7 ==> block 2, local5 <= local3
block 6 ==> block 7,
block 2 ==> block 6, local1[local5] <> local4
block 2 ==> block 5, not local1[local5] <> local4
block 0 ==> block 7,
returnblock 8 5
end
}

```

도면6



도면7

(a)	<pre> PRE ((0 <= local2) && (local3 < local1.length)) local5 := local2 LOOPINV ((local2 <= local5) && (ForAll int j. (((local2 <= j) && (j < local5)) ==> not((local1[j] == local4)))))) LOOPINV ((local2 <= local5) && (ForAll int j. (((local2 <= j) && (j < local5)) ==> not((local1[j] == local4)))))) ASSUME not((local5 <= local3)) return_value := 0 POST (result == (Exist int j. (((local2 <= j) && (j < local3)) && (local1[j] == local4)))) </pre>
(b)	
(c)	<pre> LOOPINV ((local2 <= local5) && (ForAll int j. (((local2 <= j) && (j < local5)) ==> not((local1[j] == local4)))))) ASSUME (local5 <= local3) ASSUME (local1[local5] <> local4) local5 := (local5 + 1) LOOPINV ((local2 <= local5) && (ForAll int j. (((local2 <= j) && (j < local5)) ==> not((local1[j] == local4)))))) LOOPINV ((local2 <= local5) && (ForAll int j. (((local2 <= j) && (j < local5)) ==> not((local1[j] == local4)))))) ASSUME (local5 <= local3) ASSUME not((local1[local5] <> local4)) return_value := 1 POST (result == (Exist int j. (((local2 <= j) && (j < local3)) && (local1[j] == local4)))) </pre>
(d)	

도면8

(a)	$((0 \leq \text{local2}) \ \&\& \ (\text{local3} < \text{local1.length})) \implies ((\text{local2} \leq \text{local2}) \ \&\& \ (\text{ForAll} \ \text{int} \ j. ((\text{local2} \leq j) \ \&\& \ (j < \text{local2})) \implies \text{not}((\text{local1}[j] == \text{local4}))))))$
(b)	$((\text{local2} \leq \text{local5}) \ \&\& \ (\text{ForAll} \ \text{int} \ j. ((\text{local2} \leq j) \ \&\& \ (j < \text{local5})) \implies \text{not}((\text{local1}[j] == \text{local4})))))) \implies (\text{not}((\text{local5} \leq \text{local3})) \implies (\text{result} == (\text{Exist} \ \text{int} \ j. ((\text{local2} \leq j) \ \&\& \ (j < \text{local3})) \ \&\& \ (\text{local1}[j] == \text{local4}))))))$
(c)	$((\text{local2} \leq \text{local5}) \ \&\& \ (\text{ForAll} \ \text{int} \ j. ((\text{local2} \leq j) \ \&\& \ (j < \text{local5})) \implies \text{not}((\text{local1}[j] == \text{local4})))))) \implies ((\text{local5} \leq \text{local3}) \implies ((\text{local1}[\text{local5}] \neq \text{local4}) \implies ((\text{local2} \leq (\text{local5} + 1)) \ \&\& \ (\text{ForAll} \ \text{int} \ j. ((\text{local2} \leq j) \ \&\& \ (j < (\text{local5} + 1))) \implies \text{not}((\text{local1}[j] == \text{local4})))))))$
(d)	$((\text{local2} \leq \text{local5}) \ \&\& \ (\text{ForAll} \ \text{int} \ j. ((\text{local2} \leq j) \ \&\& \ (j < \text{local5})) \implies \text{not}((\text{local1}[j] == \text{local4})))))) \implies ((\text{local5} \leq \text{local3}) \implies (\text{not}((\text{local1}[\text{local5}] \neq \text{local4}))) \implies (\text{result} == (\text{Exist} \ \text{int} \ j. ((\text{local2} \leq j) \ \&\& \ (j < \text{local3})) \ \&\& \ (\text{local1}[j] == \text{local4}))))))$

도면9

