



(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2012년10월17일
(11) 등록번호 10-1192438
(24) 등록일자 2012년10월11일

(51) 국제특허분류(Int. Cl.)
G06F 9/45 (2006.01) G06F 11/28 (2006.01)
(21) 출원번호 10-2012-0048575
(22) 출원일자 2012년05월08일
심사청구일자 2012년05월08일
(56) 선행기술조사문헌
KR100597413 B1
KR1020050075647 A
KR1020090039067 A
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 7 항

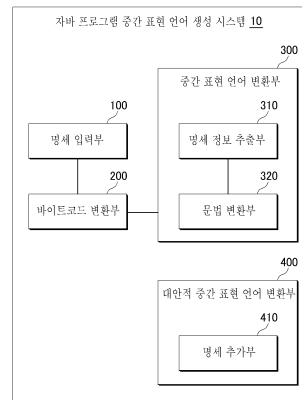
심사관 : 지정훈

(54) 발명의 명칭 자바 프로그램 중간 표현 언어 생성 시스템 및 방법

(57) 요약

프로그램 검증을 위한 명세 정보를 포함하는 자바 프로그램 소스 파일을 생성하는 명세 입력부; 상기 명세 정보가 포함되어 있는 소스 파일을 상기 명세 정보를 포함하는 바이트코드로 변환하는 바이트코드 변환부; 및 상기 명세 정보가 포함되어 있는 바이트코드를 프로그램 검증 조건을 포함하는 중간 표현 언어로 변환하는 중간 표현 언어 변환부;를 포함하는 자바 프로그램 중간 표현 언어 생성 시스템을 제공한다.

대표도 - 도1



(72) 발명자

김제민

인천광역시 부평구 십정동 181-112 2층

김선태

인천광역시 남구 용현동 31-21번지 1층

이 발명을 지원한 국가연구개발사업

과제고유번호 2011-0026495

부처명 교육과학기술부

연구사업명 일반연구자 지원사업(기본연구)

연구과제명 SMT-Solver 기반 자바 바이트코드 프로그램 검증 도구의 설계 및 구현

주관기관 인하대학교 산학협력단

연구기간 2010.09.01 ~ 2012.08.31

특허청구의 범위

청구항 1

자바 프로그램 중간 표현 언어 생성 시스템을 이용한 자바 프로그램 중간 표현 언어 생성 방법에 있어서,

- (a) 사용자로부터 자바 프로그램 중간 표현 언어의 원천에 대한 선택을 수신하는 단계;
- (b) 상기 (a) 단계의 결과가 소스일 경우, 프로그램 검증을 위한 명세 정보를 포함하는 자바 프로그램 소스 파일을 생성하는 단계;
- (c) 상기 명세 정보가 포함되어 있는 소스 파일을 상기 명세 정보를 포함하는 바이트코드로 변환하는 단계; 및
- (d) 상기 명세 정보가 포함되어 있는 바이트코드를 프로그램 검증 조건을 포함하는 중간 표현 언어로 변환하는 단계;를 포함하는 자바 프로그램 중간 표현 언어 생성 방법.

청구항 2

제 1 항에 있어서,

상기 (a) 단계의 결과가 바이트코드일 경우, 명세 정보가 포함되어 있지 않은 자바 바이트코드를 중간 표현 언어로 변환하는 단계;를 더 포함하는 자바 프로그램 중간 표현 언어 생성 방법.

청구항 3

제 1 항에 있어서,

상기 (b) 단계는 명세 정보 입력에 JML(Java Modeling Language) 문법을 사용하는 자바 프로그램 중간 표현 언어 생성 방법.

청구항 4

제 1 항에 있어서,

상기 명세 정보는 전조건, 후조건, 및 루프불변자를 포함하는 자바 프로그램 중간 표현 언어 생성 방법.

청구항 5

제 1 항에 있어서,

상기 (c) 단계는 상기 명세 정보가 포함되어 있는 소스 파일을 상기 명세 정보를 포함하는 바이트코드로 변환하는 데 JML2BML 도구를 사용하는 자바 프로그램 중간 표현 언어 생성 방법.

청구항 6

제 1 항에 있어서,

상기 (d) 단계는

상기 명세 정보를 포함하는 바이트코드로부터 명세 정보를 추출하는 단계; 및

추출한 상기 명세 정보를 상기 중간 표현 언어 문법으로 변환하는 단계;를 포함하는 자바 프로그램 중간 표현 언어 생성 방법.

청구항 7

제 1 항 내지 제 6 항 중 어느 하나의 항에 기재된 방법을 수행하는 프로그램이 기록된 컴퓨터가 읽기 가능한 기록 매체.

청구항 8

삭제

청구항 9

삭제

청구항 10

삭제

명세서

기술분야

[0001] 본 발명은 자바 프로그램 중간 표현 언어 생성 시스템 및 방법에 관한 것이다.

배경기술

[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 주석과 함께 제공되는 소스 코드를 포함하는 컴퓨터 프로그램에 대해 자동으로 비헤이비어(behavior) 패턴을 생성해주는 구성을 개시하고 있다.

발명의 내용

해결하려는 과제

[0006] 본 발명은 전술한 자바 프로그램 검증 문제를 해결하기 위한 것으로서, 그 목적은 자바 소스 또는 클래스 파일을 입력받아 명세를 추가하고 중간 표현 언어로 변환하는 자바 프로그램 중간 표현 언어 생성 시스템 및 방법을 제공하는 것이다.

과제의 해결 수단

[0007] 상기와 같은 목적을 달성하기 위한 본 발명의 제 1 측면에 따른 자바 프로그램 중간 표현 언어 생성 시스템은 자바 프로그램 중간 표현 언어 생성 시스템에 있어서, 프로그램 검증을 위한 명세 정보를 포함하는 자바 프로그램 소스 파일을 생성하는 명세 입력부; 상기 명세 정보가 포함되어 있는 소스 파일을 상기 명세 정보를 포함하는 바이트코드로 변환하는 바이트코드 변환부; 및 상기 명세 정보가 포함되어 있는 바이트코드를 프로그램 검증 조건을 포함하는 중간 표현 언어로 변환하는 중간 표현 언어 변환부;를 포함하는 것을 특징으로 한다.

[0008] 상기와 같은 목적을 달성하기 위한 본 발명의 제 2 측면에 따른 자바 프로그램 검증 방법은 프로그램 검증을 위한 명세 정보를 포함하는 자바 프로그램 소스 파일을 생성하는 단계; 상기 명세 정보가 포함되어 있는 소스 파일을 상기 명세 정보를 포함하는 바이트코드로 변환하는 단계; 및 상기 명세 정보가 포함되어 있는 바이트코드를 프로그램 검증 조건을 포함하는 중간 표현 언어로 변환하는 단계;를 포함하되, 상기 중간 표현 언어로 변환하는 단계는 상기 명세 정보를 포함하는 바이트코드로부터 명세 정보를 추출하는 단계; 및 추출한 상기 명세 정보를 상기 중간 표현 언어 문법으로 변환하는 단계;를 포함하는 것을 특징으로 한다.

발명의 효과

[0009] 본 발명에 따른 자바 프로그램 중간 표현 언어 생성 시스템 및 자바 프로그램 검증 방법은 자바 소스와 클래스

파일을 선택적으로 입력받아 명세를 추가하고 중간 표현 언어로 변환할 수 있다.

[0010] 따라서 자바 소스가 공개되어 있을 때는 고수준에서 명세를 손쉽게 추가할 수 있으며, 자바 소스를 사용할 수 없을 때는 자바 클래스 파일을 이용하여 검증할 수 있어 유연하다.

도면의 간단한 설명

[0011] 도 1은 본 발명에 따른 자바 프로그램 중간 표현 언어 생성 시스템의 구조를 도시함.

도 2, 3은 본 발명에 따른 자바 프로그램 검증 방법의 흐름을 도시함.

도 4, 5, 6은 본 발명에 따른 자바 프로그램 검증 방법의 실시예를 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 실시예를 상세히 설명한다. 그러나 본 발명은 여러 가지 상이한 형태로 구현될 수 있으며 여기에서 설명하는 실시예에 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 발명에 따른 자바 프로그램 중간 표현 언어 생성 시스템(10)을 나타낸 블록도이다.

[0015] 먼저 도 1을 참조하면, 본 발명에 따른 자바 프로그램 중간 표현 언어 생성 시스템(10)은 명세 입력부(100), 바이트코드 변환부(200), 중간 표현 언어 변환부(300)를 포함하며, 중간 표현 언어 변환부(300)는 명세 정보 추출부(310), 문법 변환부(320)를 포함한다. 또한 자바 프로그램 중간 표현 언어 생성 시스템(10)은 대안적 중간 표현 언어 변환부(400)를 더 포함할 수 있으며, 대안적 중간 표현 언어 변환부(400)는 명세 추가부(410)를 더 포함할 수 있다.

[0016] 명세 입력부(100), 바이트코드 변환부(200), 중간 표현 언어 변환부(300)는 자바 소스 파일(예: .java 파일)이 존재할 때 프로그램 검증을 위한 명세를 추가한 후 중간 표현 언어로 변환하기 위한 구성 요소이며, 대안적 중간 표현 언어 변환부(400)는 자바 소스 파일이 존재하지 않아 자바 바이트코드(예: .class 파일)를 사용하여 검증을 수행해야 할 때 클래스 파일을 중간 표현 언어로 변환하기 위한 구성 요소이다. 따라서 대안적 중간 표현 언어 변환부(400)는 추가로 명세를 추가할 때 사용할 수 있는 명세 추가부(410)를 더 포함할 수 있다. 클래스 파일(.class)에는 명세가 포함되어 있지 않을 가능성이 많기 때문이다.

[0017] 본 발명에 따른 자바 프로그램 중간 표현 언어 생성 시스템(10)은 이렇게 프로그램 소스 파일의 존재 유무에 따라 소스 파일과 클래스 파일을 선택적으로 입력받아 중간 표현 언어로 변환할 수 있기 때문에 유연하다.

[0018] 자바 소스 파일은 공개되어 있지 않고 클래스 파일만 공개되어 있는 경우가 많기 때문에, 사용자는 많은 경우 대안적 중간 표현 언어 변환부(400)를 사용하여 자바 프로그램을 중간 표현 언어로 변환하여 검증할 수 있다. 그러나 그럴 경우, 사용된 클래스 파일에는 전조건, 후조건, 루프불변자와 같은 명세들을 포함하고 있지 않을 가능성이 많다. 따라서 검증을 수행하기 위해서는 사용자가 직접 명세 정보를 중간 표현 언어에 추가해 주어야 한다.

[0019] 이렇게 저수준에서 중간 표현 언어에 직접 명세를 추가하는 것보다는, 고수준에서 즉, 자바 소스 파일에 명세를 추가하는 것이 사용자에게 훨씬 편리하다. 사용자는 주석을 사용하여 프로그램의 행동(behavior)을 기술하는 명세 정보를 소스에 추가할 수 있으므로 훨씬 직관적이고 쉽다.

[0020] 즉, 본 발명에 따른 자바 프로그램 중간 표현 언어 생성 시스템(10)은 두가지 옵션을 제공하여 사용자가 상황에 따라 좀더 효율적인 방법을 선택할 수 있고 효율적인 방법을 사용할 수 없을 때는 대안을 선택할 수 있도록 한다. 다시 말해, 본 발명에 따른 자바 프로그램 중간 표현 언어 생성 시스템(10)은 명세 정보가 포함되어 있지 않은 자바 바이트코드(예: .class 파일)와 명세 정보가 포함되어 있는 자바 바이트코드(예: 바이트코드 변환부(200)의 출력) 중에서 중간 표현 언어를 생성하는 데 사용할 입력을 선택할 수 있는 옵션을 사용자에게 제공한

다.

- [0021] 본 명세서에서는 이중에서 소스 파일에 명세를 추가하여 중간 표현 언어로 변환하는 데 사용되는 명세 입력부(100), 바이트코드 변환부(200), 및 중간 표현 언어 변환부(300)를 집중적으로 설명한다.
- [0022] 명세 입력부(100)를 통해, 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.
- [0023] 이러한 명세 정보는 명세 언어를 통해 기술된다. 예컨대, 본 발명에 따른 자바 프로그램 중간 표현 언어 생성 시스템(10)은 JML(Java Modeling Language)를 사용할 수 있다. 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.
- [0024] 전조건, 후조건, 및 루프불변자 등의 명세 정보나 JML 등의 명세 언어는 종래 기술이므로 자세한 설명을 생략한다.
- [0025] 명세 입력부(100)를 통해 생성된 명세 정보를 포함하는 소스 파일은 바이트코드 변환부(200)를 통해 명세 정보를 포함하는 바이트코드(bytecode)로 변환된다. 명세 정보를 포함하는 바이트코드는 예컨대 BML(Bytecode Modeling Language) 형태의 바이트코드일 수 있으며, 이때, 본 발명에 따른 자바 프로그램 중간 표현 언어 생성 시스템(10)의 바이트코드 변환부(200)는 JML(명세 정보가 포함되어 있는 소스 파일)을 BML(명세 정보가 포함되어 있는 바이트코드)로 변환하기 위해 JML2BML이라는 도구를 사용할 수 있다. 이 도구는 종래 기술이므로 자세한 설명을 생략한다.
- [0026] 중간 표현 언어 변환부(300)는 명세 정보가 포함되어 있는 바이트코드를 프로그램 검증 조건을 포함하는 중간 표현 언어로 변환한다. 예컨대, 본 발명에 따른 자바 프로그램 중간 표현 언어 생성 시스템(10)의 중간 표현 언어 변환부(300)는 BIRS(Bytecode Intermediate Representation Specification)이라는 중간 표현 언어를 출력할 수 있다.
- [0027] 이 중간 표현 언어는 역 컴파일을 통해 자바 바이트코드로부터 자바 소스를 얻는다고 해도 컴파일과정 중 손실되는 정보들을 완전히 복구할 수는 없기 때문에, 자바 바이트코드만으로 해당 프로그램을 검증해야 할 필요에 의해 본 발명의 발명자에 의해 개발되었으며, 스택 기반의 자바 프로그램을 스택리스 형태로 변환하여 검증하는 등의 특징을 가지고 있다. 역시 종래 기술이므로 자세한 설명은 생략한다.
- [0028] 중간 표현 언어 변환부(300)는 명세 정보 추출부(310)와 문법 변환부(320)를 포함한다.
- [0029] 명세 정보 추출부(310)는 명세 정보가 포함되어 있는 바이트코드(예: BML)로부터 명세 정보를 추출하며, 문법 변환부(320)는 추출된 명세 정보를 목표 중간 표현 언어(예: BIRS)에 맞게 변환한다. 자세한 내용은 실시예를 통해 후술한다.
- [0030] 도 2는 본 발명에 따른 자바 프로그램 검증 방법의 흐름을 도시하고 있다.
- [0031] 전술한 바와 같이 자바 소스 파일과 자바 클래스 파일 중에서 사용할 입력을 선택할 수 있기 때문에, 먼저 입력이 .class 파일인지 판단한다(S210). 입력이 클래스 파일이면 이를 바로 중간 표현 언어로 변환한다(S240).
- [0032] 입력이 소스 파일이면 명세 정보를 추가하여 명세를 포함하는 소스를 생성한다(S220).
- [0033] 다음 이를 명세를 포함하는 바이트코드로 변환한다(S230).
- [0034] 다음 이를 중간 표현 언어로 변환한다(S240).
- [0035] 도 3은 입력이 소스 파일일 경우의 상기 S240 단계의 흐름을 도시하고 있다.
- [0036] 명세 정보가 포함되어 있는 바이트코드(예: BML)로부터 명세 정보를 추출한다(S310).
- [0037] 다음, 추출된 명세 정보를 목표 중간 표현 언어(예: BIRS)에 맞게 변환한다(S320).
- [0038] 도 4, 5, 6은 본 발명에 따른 자바 프로그램 검증 방법의 실시예를 도시하고 있다. 도 4는 JML로 기술된 자바 소스, 도 5는 BML로 변환된 바이트코드, 그리고 도 6은 BIRL 중간 표현 언어로 변환된 모습이다.

- [0039] 도 4를 보면, 간단한 검색을 수행하는 메소드 linearSearch를 갖고 있는 클래스 Search가 도시되어 있다. 전술한 바와 같이, @로 시작하는 주석이 명세 기술 언어 JML로 기술한 명세 정보이다.
- [0040] linearSearch 메소드가 실행되기 전 만족해야 하는 성질을 나타내기 위해 requires 절이 기술되어 있다(도면에서 //@ requires 0<=l && u<a.length; 부분). 여기서 requires 절은 linearSearch 메소드가 실행하기 전에 인수인 l이 0보다 크거나 같고 인수인 u가 배열 a의 길이보다 작아야 함을 의미한다.
- [0041] 메소드는 또한 메소드가 실행된 후 만족해야 하는 성질을 ensures 절로 나타낸다(도면에서 //@ensures esult == (Wexists int j; l<=j && j<=u && a[j]==e); 부분). ensures 절에서 esult는 linearSearch 메소드가 실행된 후 반환 값을 나타내며, Wexists int j; l<=j && j<=u && a[j]==e는 술어(predicate)로서 배열 a의 인덱스 경계를 넘지 않는 범위 내에서 인수 e와 같은 원소가 배열 a에 존재함을 나타낸다. 따라서 linearSearch 메소드의 ensures절은 반환값이 배열 a에 인수 e와 같은 원소가 존재하는지 여부와 동일함을 나타낸다.
- [0042] linearSearch 메소드 안에 for 루프가 존재하는데 loop_invariant 절을 통해 for 루프가 실행될 때마다 만족해야 하는 성질을 나타낼 수 있다(도면에서 //@ loop_invariant l<=i && (Wforall int j; l<=j && j<i ==> a[j]!=e); 부분). 여기서 loop_invariant 절이 나타내는 바는 for 루프가 실행될 때마다 인덱스 변수 i는 인수 l보다 크거나 같아야 하고 l과 i 사이의 값 j로 가리켜지는 배열 a의 원소는 인수 e와 같지 않아야 한다는 것이다.
- [0043] 도면의 소스 파일은 JML로 loop 불변자, ensures, requires를 나타내었고, 배열 객체의 필드 접근, 배열의 원소 접근, 논리 연산자인 and, implication, forall, exists, logical equivalence 등으로 논리식이 구성된다.
- [0044] javac 컴파일러를 통해 Search.java 파일을 컴파일하게 되면 자바 가상 머신 명세에 맞게 .class 파일이 생성된다. 클래스 파일은 아래와 같은 형태로 구성된다.
- [0045] ClassFile {
- [0046] u4 magic;
- [0047] u2 minor_version;
- [0048] u2 major_version;
- [0049] u2 constant_pool_count;
- [0050] cp_info constant_pool[constant_pool_count-1];
- [0051] u2 access_flags;
- [0052] u2 this_class;
- [0053] u2 super_class;
- [0054] u2 interfaces_count;
- [0055] u2 interfaces[interfaces_count];
- [0056] u2 fields_count;
- [0057] field_info fields[fields_count];
- [0058] u2 methods_count;
- [0059] method_info methods[methods_count];
- [0060] u2 attributes_count;
- [0061] attribute_info attributes[attributes_count];
- [0062] }
- [0063] 하지만 도 4에 도시되어 있는 명세 정보는 자바 가상 머신 명세에서 나타내는 클래스 파일 포맷에는 포함되어 있지 않다.

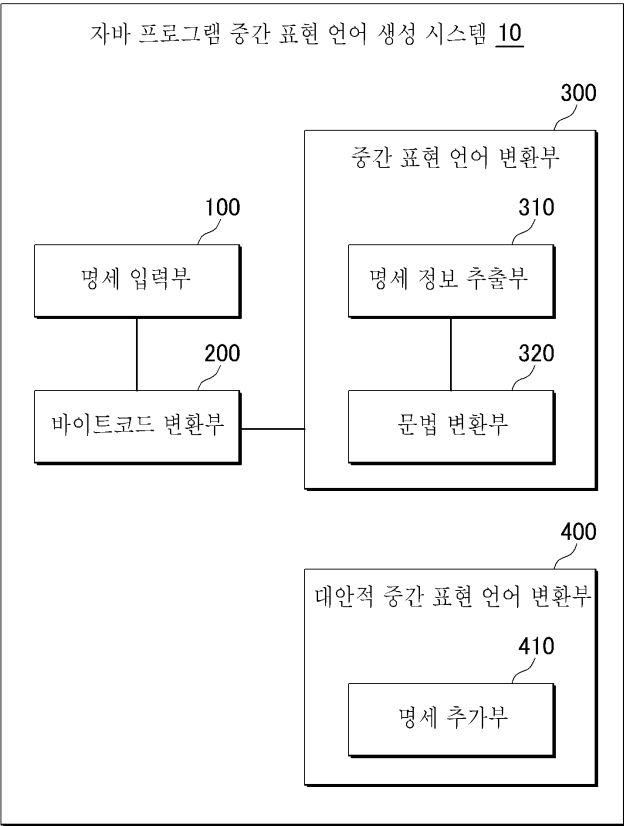
- [0064] 도 5는 도 4의 소스가 변환된 BML이 도시되어 있다. 변환에는 JML2BML 도구가 사용되었다. BML 코드는 기본적으로 자바 가상 머신 명세를 따른다. 즉, BML은 자바 바이트 코드에서 확장된 언어이다. 자바 바이트 코드를 역 컴파일 시 사용되는 명령어인 Javap 명령어를 통해 BML을 역 컴파일 했을 때 전조건, 후조건, 루프불변자 등의 명세정보가 출력되지 않는다.
- [0065] BML 코드만의 특징은 클래스 파일의 포맷 중 attributes 필드에 삽입하게 된다. 예컨대, BML은 자바 가상 머신 명세에는 존재하지 않는 2차 상수 풀(constant pool) 등을 포함한다. 이는 BML의 2차 상수풀을 통해 명세 정보를 포함할 수 있음을 의미한다.
- [0066] 도 6은 도 5의 BML을 입력받아 출력된 BIRS가 도시되어 있다. 진한 글씨체로 표현된 것이 BML에 포함되어 있는 명세 정보가 BIRS에 반영된 것이다.
- [0067] 전술한 바와 같이 BML의 자바 가상 머신 명세에는 존재하지 않는(즉, 일반적인 .class 파일에는 존재하지 않는) 구성 요소를 사용하여 명세 정보를 포함시킬 수 있고, 이 구성 요소로부터 명세 정보를 추출할 수 있다. 명세 정보 추출부(310)가 이러한 역할을 담당한다.
- [0068] 예컨대, BML 클래스 파일 안에 정의되어 있는 메소드는 속성 정보를 가지고 있다. 속성은 속성 이름 인덱스 정보를 가지고 있고, 이 정보를 통해 상수 풀에 정의되어 있는 이름 타입과 값을 알 수 있다. 이름 타입과 값 정보를 가지고 BML 내부에 정의되어 있는 명세 정보 스펙과 일치하는 정보를 찾아 반환한다. 반환되는 정보를 분석하면 전조건, 후조건, 루프불변자 등의 명세 정보에 해당하는 이진 코드를 찾을 수 있다.
- [0069] 획득한 명세 정보를 BIRS의 문법에 따라 변환하는 역할은 문법 변환부(320)가 담당한다. 명세 정보를 중간 표현 언어의 문법에 맞게 변환하는 것이다. 이러한 과정은 당업자에게 명백한 것이므로 설명을 생략한다.
- [0070] 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 실시예들은 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.
- [0071] 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

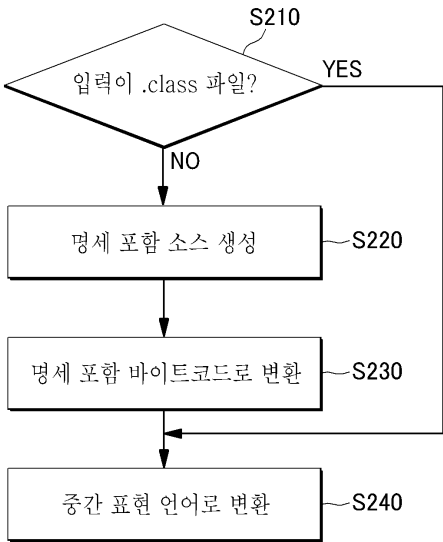
- [0072] 10: 자바 프로그램 중간 표현 언어 생성 시스템
- 100: 명세 입력부
- 200: 바이트코드 변환부
- 300: 중간 표현 언어 변환부

도면

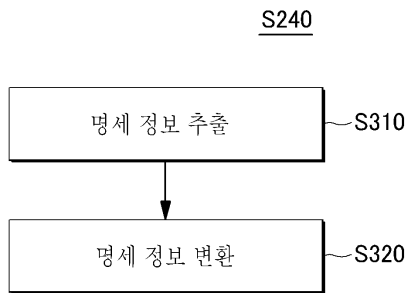
도면1



도면2



도면3



도면4

```

package main;

public class Search {

    //@ requires 0 <= l && u < a.length;
    //@ ensures \result == (\exists int j; l <= j && j <= u && a[j] == e);
    public boolean linearSearch(int[] a, int l, int u, int e){

        //@ loop invariant l <= i && (\forall int j; l <= j && j < i => a[j] != e);
        for(int i=l; i <= u; i=i+1){
            if(a[i]==e)
                return true;
        }
        return false;
    }
}
    
```

도면5

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
00000000h:	CA	FE	BA	BE	00	00	00	32	00	22	07	00	02	01	00	0B	; Ēp¼...2.".....
00000010h:	6D	61	69	6E	2F	53	65	61	72	63	68	07	00	04	01	00	; main/Search.....
00000020h:	10	6A	61	76	61	2F	6C	61	6E	67	2F	4F	62	6A	65	63	; .java/lang/Objec
00000030h:	74	01	00	06	3C	69	6E	69	74	3E	01	00	03	28	29	56	; t...<init>...()V
00000040h:	01	00	04	43	6F	64	65	0A	00	03	00	09	0C	00	05	00	; ...Code.....
00000050h:	06	01	00	0F	4C	69	6E	65	4E	75	6D	62	65	72	54	61	; LineNumberTa
00000060h:	62	6C	65	01	00	12	4C	6F	63	61	6C	56	61	72	69	61	; ble...LocalVaria
00000070h:	62	6C	65	54	61	62	6C	65	01	00	04	74	68	69	73	01	; bleTable...this.
00000080h:	00	0D	4C	6D	61	69	6E	2F	53	65	61	72	63	68	3B	01	; ..Lmain/Search;
00000090h:	00	0C	6C	69	6E	65	61	72	53	65	61	72	63	68	01	00	; ..linearSearch..
000000a0h:	08	28	5B	49	49	49	49	29	5A	01	00	01	61	01	00	02	; .([IIII)Z...a...
000000b0h:	5B	49	01	00	01	6C	01	00	01	49	01	00	01	75	01	00	; [I...l...I...u...
000000c0h:	01	65	01	00	01	69	01	00	0D	53	74	61	63	6B	4D	61	; .e...i...StackMa
000000d0h:	70	54	61	62	6C	65	01	00	0A	53	6F	75	72	63	65	46	; pTable...SourceF
000000e0h:	69	6C	65	01	00	0B	53	65	61	72	63	68	2E	6A	61	76	; ile...Search.jav
000000f0h:	61	01	00	03	69	6E	74	01	00	18	6F	72	67	2E	62	6D	; a...int...org.bm
00000100h:	6C	73	70	65	63	73	2E	41	73	73	65	72	74	54	61	62	; lsspecs.AssertTab
00000110h:	6C	65	01	00	17	6F	72	67	2E	62	6D	6C	73	70	65	63	; le...org.bmlspec
00000120h:	73	2E	49	6E	76	61	72	69	61	6E	74	73	01	00	20	6F	; s.Invariants.. o
00000130h:	72	67	2E	62	6D	6C	73	70	65	63	73	2E	4D	65	74	68	; rg.bmlspecs.Meth
00000140h:	6F	64	53	70	65	63	69	66	69	63	61	74	69	6F	6E	01	; oSpecification.
00000150h:	00	1F	6F	72	67	2E	62	6D	6C	73	70	65	63	73	2E	53	; ..org.bmlspecs.S
00000160h:	65	63	6F	6E	64	43	6F	6E	73	74	61	6E	74	50	6F	6F	; econdConstantPoo
00000170h:	6C	01	00	23	6F	72	67	2E	62	6D	6C	73	70	65	63	73	; l..#org.bmlspecs
00000180h:	2E	4C	6F	6F	70	53	70	65	63	69	66	69	63	61	74	69	; ..LoopSpecificati
00000190h:	6F	6E	54	61	62	6C	65	01	00	1B	6F	72	67	2E	62	6D	; onTable...org.bm
000001a0h:	6C	73	70	65	63	73	2E	46	69	65	6C	64	4D	6F	64	69	; lsspecs.FieldModi
000001b0h:	66	69	65	72	73	01	00	18	6F	72	67	2E	62	6D	6C	73	; fiers...org.bmls
000001c0h:	70	65	63	73	2E	47	68	6F	73	74	46	69	65	6C	64	73	; pecks.GhostFields
000001d0h:	00	21	00	01	00	03	00	00	00	00	02	00	01	00	05	;	!.....
000001e0h:	00	06	00	01	00	07	00	00	00	2F	00	01	00	01	00	00	;/.....
000001f0h:	00	05	2A	B7	00	08	B1	00	00	00	02	00	0B	00	00	00	; ..*...±.....
00000200h:	0C	00	01	00	00	05	00	0C	00	0D	00	00	00	0A	00	;	
00000210h:	00	00	06	00	01	00	00	03	00	01	00	0E	00	0F	00	;	
00000220h:	04	00	1D	00	00	45	00	01	02	13	40	00	00	00	00	;	E.....@....
00000230h:	90	00	02	12	90	00	03	63	90	00	01	80	00	26	00	01	; C.....&..
00000240h:	D0	10	52	0B	01	C0	00	1A	00	27	02	02	13	90	00	02	; D.R..A...'.....
00000250h:	E0	00	28	13	E0	00	28	90	00	03	10	61	90	00	01	E0	; à.(.à.(.....a...à
00000260h:	00	28	90	00	04	00	01	00	29	00	00	00	00	1B	00	00	; (.).....
00000270h:	00	02	00	00	00	1F	00	00	00	35	00	01	00	17	00	00	; 5.....
00000280h:	02	13	90	00	02	90	00	05	0A	01	C0	00	1A	00	27	04	; A...'.
00000290h:	02	13	90	00	02	E0	00	28	12	E0	00	28	90	00	05	17	; à.(.à.(.....
000002a0h:	61	90	00	01	E0	00	28	90	00	04	40	00	00	00	01	00	; a.....à.(.....@.....
000002b0h:	07	00	00	00	96	00	02	00	06	00	00	00	1C	1C	36	05	; 6.....
000002c0h:	A7	00	11	2B	15	05	2E	15	04	A0	00	05	04	AC	84	05	; \$.+.....~.
000002d0h:	01	15	05	1D	A4	FF	EF	03	AC	00	00	00	03	00	17	00	; ÿï.~.....
000002e0h:	00	00	08	00	03	FC	00	06	01	0A	02	00	0B	00	00	00	; ü.....
000002f0h:	3E	00	06	00	00	00	1C	00	0C	00	0D	00	00	00	00	00	; >.....
00000300h:	1C	00	10	00	11	00	01	00	00	00	1C	00	12	00	13	00	;>.....
00000310h:	02	00	00	00	1C	00	14	00	13	00	03	00	00	00	1C	00	;>.....
00000320h:	15	00	13	00	04	00	03	00	18	00	16	00	13	00	05	00	;>.....
00000330h:	0A	00	00	00	16	00	05	00	00	00	0A	00	06	00	0B	00	;>.....
00000340h:	0F	00	0C	00	11	00	0A	00	1A	00	0E	00	03	00	18	00	;>.....
00000350h:	00	00	02	00	19	00	1C	00	00	00	02	00	00	00	1E	00	;>.....
00000360h:	00	00	41	00	21	00	08	01	00	05	69	6E	74	5B	5D	07	; ..A!.....int[.]
00000370h:	00	22	01	00	06	6C	65	6E	67	74	68	0C	00	24	00	11	; .."....length..\$.
00000380h:	09	00	23	00	25	01	00	01	6A	0C	00	27	00	13	01	00	; ..#.%...j...'....
00000390h:	13	6A	61	76	61	2F	6C	61	6E	67	2F	45	78	63	65	70	; .java/lang/Excep
000003a0h:	74	69	6F	6E													; tion

도면6

```

Class main.Search: (java.lang.Object ) () ()
{computationalFunction linearSearch( java.lang.Object local0, arrint local1, int local2, int local3, int local4 ) -> bool
require 0 <= local2 && local3 < local1.undef
ensure result == Exist int j. local2 <= j && j <= local3 && local1 ArrayLoad j == local4
block 8 is
8: vreturn 0
end
block 7 is
7: loopInvariant local2 <= local5 && ForAll int j. local2 <= j && j < local5 imply not local1 ArrayLoad 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: assert Not NullPointer local1
3: assert Not ArrayBound (local1, local5)
4: ifd(local1 ArrayLoad local5 <> local4) 6
end
block 0 is
0: assignVal local5 := local2
1: goto 7
end
block 7 ==> block 8, not i:I<=u:I
block 7 ==> block 2, i:I<=u:I
block 6 ==> block 7,
block 2 ==> block 6, a:O[i:I]:I<>e:I
block 2 ==> block 5, not a:O[i:I]:I<>e:I
block 0 ==> block 7,
returnblock 8 5
end
computationalFunction <init>( java.lang.Object local0 ) -> unit
require true
ensure true
block 0 is
0: assert Not NullPointer local0
1: invokeNonVirtual local0.java.lang.Object.<init>
2: return
end
returnblock 0
end
}

```