제36회 한국정보처리학회 추계학술발표대회 논문집 제18권 제2호 (2011.11)

프로그램 검증을 위한 중간표현 언어의 분석

김선태*, 김제민*, 박준석*, 유원희*
*인하대학교 컴퓨터정보공학과
e-mail: kst_1205@nate.com

Analysis of the intermediate representation langauge for program verification

SeonTae Kim*, JeMin Kim*, JoonSeok Park*, WeonHee Yoo*
*Dept of Computer and Information Engineering, Inha University

요 약

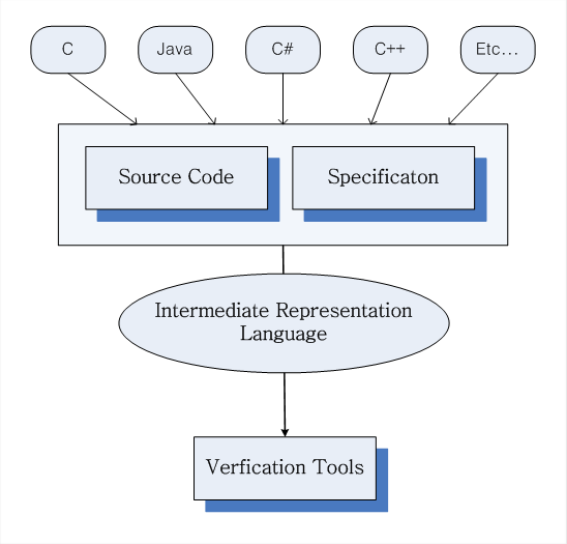
소프트웨어의 비중이 커짐에 따라 소프트웨어가 안전하게 실행되는 것이 보장되어야 한다. 이를 위해 다양한 검증 도구를 통해 검증이 수행된다. 하지만 소스 코드와 명세를 입력으로 받는 검증도구는 검증조건 생성이 어렵기 때문에 검증 조건 생성에 용이하도록 입력 값을 중간 표현 언어로 변환해 주는 것이 필요하다. 본 논문에서는 검증의 정확성을 위해 다양하게 존재하는 중간 표현 언어의 특성을 분석하고 예제를 통해 비교한다. 중간 표현 언어의 비교 분석 결과를 통해 검증을 수행할 때 검증의 목적과 환경에 적합한 중간 표현 언어 선택으로 검증의 효율성과 정확성을 향상시킨다.

1. 서론

전 세계적으로 모든 산업에서 소프트웨어가 차지하는 비중과 중요성을 증대되어가고 있다. 과거 하드웨어의 기능을 소프트웨어가 대체하는 방향으로 발전하고 있다. 임베디드 시스템 및 유비쿼터스 컴퓨팅으로 대별되는 미래의 IT 환경에서는 소프트웨어의 비중과 중요성이 증가할 것이다. 특히, 항공, 우주, 핵발전소 등 첨단 산업 분야에 사용되는 소프트웨어의 중요성과 비중은 훨씬 크다. 소프트웨어의 비중과 중요성이 높아짐에 따라 안전성과 보안성이 검증되지 않은 소프트웨어를 실행하여 발생하는 오류는 큰 위험을 초래한다. 위험을 막기 위해서 소프트웨어가 오류 없이 안전하게 실행된다는 것을 실행 전에 보장해주는 것이 필요하다.

소프트웨어의 안전한 실행 보장의 필요성으로 인해 현재 소프트웨어의 검증 도구는 다양하게 존재한다. 검증도구의 입력 형태는 그림 1과 같이 이루어져 있다. 검증하고자 하는 소스코드와 검증하고자 하는 성질을 나타내는 명세가 검증 도구의 입력 값으로 들어간다. 검증 도구는 검증 조건을 생성하고 정리증명기(Theorem Prover)를 통해 증명한다. 증명 결과를 통해 소프트웨어의 오류 여부를 판단한다.

하지만 검증도구의 입력으로 들어가는 소스코드와 명세는 프로그래밍 언어 다양성 등의 이유로 검증조건을 생성하기 어렵다. 이를 해결하기 위해 다양한 프로그래밍 언어를 검증 조건 생성에 용이하도록 소스코드와 명세를 포함하는 새로운 형태의 중간 표현 언어가 필요하다. 또한 중간 표현 언어는 다양한 검증 도구에 적용이 가능하도록 변환에 용이하다.



(그림 1) 검증도구의 입력 구조

현재 중간 표현 언어는 BoogiePL[1], Why[2], BIRS (Bytecode Intermediate Representation Specification)[3]가 존재한다. 각각의 중간 표현 언어는 다양한 입력 언어를 통해 생성되며 서로 다른 검증 도구를 통해 검증 조건을 생성한다. 그렇기 때문에 검증의 목적에 맞는 정확한 검증을 위해서는 중간 표현 언어의 특성을 정확하게 이해해야 한다. 하지만 현재까지 중간 표현 언어의 특징에 대한 비교는 이루어지지 않았다.

본 논문에서는 중간 표현 언어의 특성을 분석하고 예제를 통해 비교한다. 논문의 구성은 2장에서 관련연구와 3장에서 중간 표현 언어의 대한 특징, 4장에서는 비교 및 평가 마지막 5장에서는 결론과 향후 연구 과제에 관해 논의할 것이다.

2. 관련연구

2.1 정적 검증(Static program analysis)[4]

소프트웨어의 안전성을 검증하기 위한 방법으로 프로그램이 실행하기 전에 존재하는 소스코드(Source Code)와 오브젝트 코드(Object Code)를 통해 검증이 이루어진다. 정적 검증은 검증 범위와 자동화 정도에 따라 타입검사, 프로그램 검증, 확장된 정적 검증(Extended Static Checking)으로 나눌 수 있다[5].

2.2 BoogiePL[1]

BoogiePL은 자바 프로그램과 같은 객체 지향 프로그램 검증을 위해 생성된 중간 표현 언어이다. Spec#[6], BML[7] 등을 입력으로 받아 BoogiePL 형태로 바꾸어 준다. 생성된 BoogiePL은 Boogie Pipeline[8]에 따라 정리증명기를 통해 검증을 수행한다.

2.3 Why[2]

Why는 입력 언어의 제한 없이 주석 표시된 프로그램(Annotated Program)을 입력으로 받아 Why의 내부 언어(Internal Language, WL)로 변환한다. 명령형 특징을 가지고 있으며 다양한 검증 도구를 통해 검증을 수행한다.

2.4 BIRS[3]

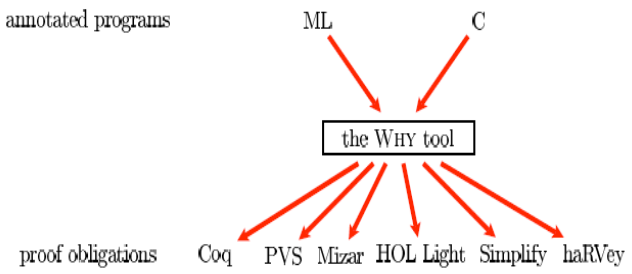
BIRS는 자바 프로그램 검증만을 위한 중간 표현 언어로서 자바 바이트 코드를 입력받아 생성한다. 스택 코드로 이루어진 자바 바이트 코드의 문제점을 해결하기 위해 스택리스 코드로 이루어져 검증에 용이하도록 설계되었다[3].

3. 중간 표현 언어의 특징

3.1 Why

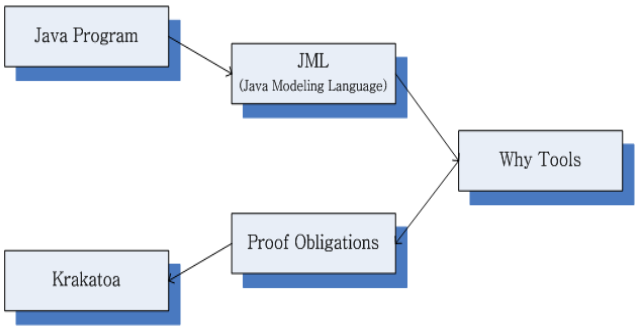
Why는 앞서 설명한 것처럼 모든 입력 언어에 제한 없이 중간 표현 언어로 변환한다. Why는 기본 데이터 타입과 추상 데이터 타입, 검증 조건 생성을 위해 필요한 주석 표시(Annotated)를 제공한다. 또한 검증에 필요한 검증 조건을 생성을 위해 Require, Ensure, Invariant를 정의하여 표현한다.

Why는 중간 표현 언어 중 프로그래머가 판독하기 쉬운 고급 언어 형태로 표현되어 있다. [2]의 예제 코드인 Index 함수의 변환과정을 참조하여 살펴보면 C언어의 의사 코드와 비슷한 형태로 나타나 있는 것을 알 수 있다.



(그림 2) Why 도구

그림 2는 Why 도구의 모습을 나타낸다. 그림 2에 나타난 것처럼 주석 표시된 ML타입의 언어와 C언어 프로그램을 입력으로 받는다. 그리고 다양한 형태의 증명 보조와 검증 도구들을 생성한다. 그림 2의 구조는 Why의 장점과 단점을 명확히 표현한다. 장점은 모든 언어를 검사하기 때문에 검증 조건을 생성하는데 유연하고, 다양한 검증 조건을 생성하기 때문에 여러 시스템에 적용하여 검증을 수행할 수 있다. 단점은 하나의 언어만을 위한 중간 표현 언어가 아니기 때문에 검증을 위한 정보가 부족할 수 있다. 그러므로 검증을 수행했을 시 검증의 정확도가 하나의 언어에 특화된 중간 표현 언어로 검증하는 것 보다 낮다. 또한 검증 단계가 복잡하여 검증을 하는데 많은 시스템을 사용해야 한다. 그림 3은 자바 프로그램을 Why를 이용하여 검증했을 때의 모습이다. Why를 통해 생성된 Proof Obligation는 Krakatoa [9]를 이용하여 검증이 수행된다.



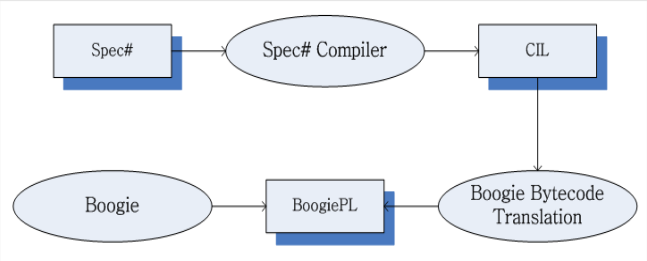
(그림 3) Why와 Krakatoa를 이용한 자바 프로그램 검증

3.2 BoogiePL

BoogiePL은 자바 프로그램과 같은 객체 지향 프로그램 검증을 목적으로 표현된 중간 표현 언어이다. BoogiePL은 C# 언어에 명세를 추가한 형태인 Spec#이 컴파일 후 생성된 바이트 코드를 입력으로 받는다. 또한, JML(Java Modeling Language)[10]을 컴파일하여 생성되는 BML(Bytecode Modeling Language)[7]을 입력으로 받아 생성한다.

BoogiePL은 프로그램 모든 명령어 표현을 정의하여 표

현하며, Why와 마찬가지로 검증조건 생성을 위해 Require, Ensure, Invariant, Assert, Assume을 정의한다. 또한 프로그램의 정보 흐름을 파악할 수 있도록 반복문과 조건문에 이름을 붙여 표현할 수 있도록 정의하였고, 분석에 용이하도록 Back edge를 havoc을 사용하여 제거하였다[11].

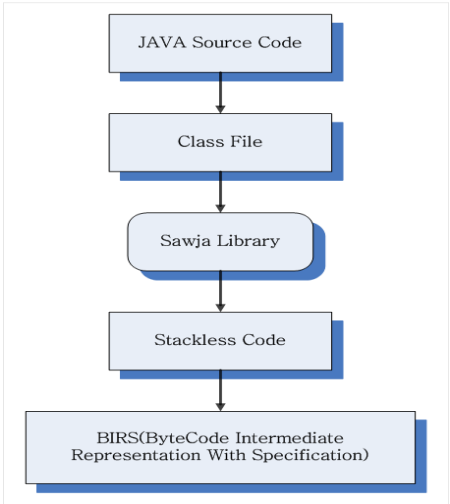


(그림 4) Boogie pipeline

Why와 비교하여 객체 지향 프로그램에 특화하여 표현되어 검증의 정확성과 정보 표현양이 향상되었다. 또한 자바 프로그램을 검증할 때 그 과정이 훨씬 간단해졌다는 것을 그림 4의 Boogie의 동작 모습으로 확인할 수 있다. Why는 3 종류의 시스템을 이용하고 있지만, Boogie는 2 종류의 시스템을 이용하고 있다. 하지만 BoogiePL은 스택기반 자바 프로그램 검증할 때 생기는 문제점[3]을 해결하지 못하였다.

3.3 BIRS

BIRS는 자바 프로그램 검증에 특화하여 생성된 중간 표현 언어이다. 자바 바이트 코드를 입력으로 받아 중간 표현 언어가 생성된다. 스택 기반 자바의 문제를 해결하기 위해 스택리스 코드로 설계되었다. 또한 자바에서 표현되는 모든 명령어와 클래스 계층 구조, CFG(Control Flow Graph)등 검증에 필요한 모든 정보를 가지고 있다.



(그림 5) BIRS 생성

그림 5는 BIRS의 생성 단계를 나타낸 모식도이다. Sawja Library[12]를 이용하여 스택기반의 바이트 코드를 스택 리스 코드로 변경하여 중간표현 언어를 생성한다.

BIRS는 다른 언어를 표현하지 못하는 단점이 존재한다. 또한 검증을 위해서는 명세 언어로의 확장이 필요하다. 하지만 자바를 검증할 때 적은 자원을 이용하여 검증이 가능하며, 더 많은 정보 생성으로 검증의 정확도가 높다.

4. 비교 및 평가

Why와 BoogiePL 그리고 BIRS을 비교하기위해 [2]와 [8] 그리고 [3]에서 나타나는 예제 프로그램을 분석한다. 각각의 중간 표현 언어는 입력 언어가 다르기 때문에 입력언어는 자바로 통일한다. 분석은 중간 표현 언어 생성이 가장 간단한 BIRS를 기준으로 수행한다.

표 1, 표 2는 명령어의 수와, 검증 표현의 수 그리고 검증 정보의 수를 중간 표현 언어와 비교하여 나타내었다.

표 1은 [3], [8]에서 공통으로 나타난 예제 코드의 비교 모습이다. BIRS는 BoogiePL보다 많은 정보량을 표현한다. 또한 BoogiePL에서 나타나는 정보는 모두 BIRS에서 나타난다.

<표 1> BoogiePL과 BIRS의 정보량 비교

	BoogiePL	BIRS
명령어 표현	7	20
정보표현	2	3
검증정보	4	6
합계	13	30

표 2는 [2]의 인덱스 Why와 [2] 코드를 BIRS로 변환한 인덱스 BIRS와의 비교 모습이다. BIRS는 Why보다 많은 정보량을 표현한다. 또한 Why에서 나타나는 정보는 BIRS에서 나타난다.

<표 2> Why와 BIRS의 정보량 비교

	Why	BIRS
명령어 표현	5	9
정보표현	2	8
검증정보	3	4
합계	10	21

표 1과 표 2의 결과는 BIRS가 BoogiePL과 Why보다 많은 양의 정보를 가지고 있음을 알 수 있다. 즉, 검증을 수행했을 때의 정확도가 높은 확률로 BIRS가 Why와 BoogiePL보다 높다는 것을 의미한다.

표 3은 BoogiePL과 Why 그리고 BIRS의 입력 언어, 검증 도구, 검증 단계를 나타낸 표이다. 입력 언어는 중간 표현 언어로 변환이 가능한 언어를 나타내고, 검증 도구는 생성된 중간 표현 언어가 이용할 수 있는 검증 도구의 종류를 나타낸다. 검증 단계는 중간 표현 언어가 검증을 수행할 때 사용하는 시스템의 개수를 나타낸다.

표 3의 결과를 통해 입력 언어의 유연성은 Why가 가장 높다. 검증의 효율성은 검증 단계가 가장 작은 BIRS가 높다. 중간 표현 언어의 유연성 역시 다양한 검증 도구 이용이 가능한 Why가 높다.

<표 3> BoogiePL, Why, BIRS 비교

	BoogiePL	Why	BIRS
입력 언어	객체 지향 언어	ML 타입 언어, C언어	자바
검증 도구	Boogie	Coq, PVS, Mizar, HOL Light, Simplify, haRVey	JBVF
검증 단계	2	3	1

5. 결론 및 향후과제

프로그램이 안전하게 실행을 보장하기 위해 다양한 검증 도구가 존재한다. 각각의 검증 도구는 검증이 용이하도록 소스코드와 명세를 포함하는 새로운 형태의 중간 표현 언어로 변환한다. 생성된 중간 표현 언어는 서로 다른 특성을 가지고 있다. 본 논문에서는 Why, BoogiePL, BIRS의 중간 표현 언어의 특성에 대해 비교 분석하였다. 비교 분석 결과를 통해 구현 언어와 구현 환경에 따라 적합한 중간 표현 언어의 선택으로 검증의 정확성과 효율성을 향상시킨다.

향후 연구과제로 중간 표현 언어를 통해 검증을 수행하는 검증 도구들의 분석을 통해 정보의 유효성을 평가한다. 정보 유효성 평가의 결과를 프로그램의 사이즈와 환경에 따라 시각화하여 표현한다. 이를 통해 검증의 효율성과 정확성을 향상시킨다.

논문 사사(acknowledgement)

이 논문은 2011년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임(No. 2010-0025660)

참고문헌

[1] Robert DeLine, K. Rustan M. Leino, “BoogiePL : A Typed procedural language for checking object-oriented program″ Technical Report MSR-TR-2005-70, 27 May 2005.

[2] Jean-Christophe Filliatre, “Why : a multi-language multi-prover verification tool″ LRI-CNRS UMR 8623, Université Paris Sud, March 2003.

[3] 김선태, 김제민, 박준석, 유원희, “자바 프로그램 검증을 위한 스택 리스 중간 표현 언어 생성기 구현”, 한국정보기술학회, 2011년 9월 30일.

[4] Static program analysis, http://en.wikipedia.org/.

[5] James, P.R. and Chalin, P., “ESC4: a modern caching ESC for Java,” In Proceedings of the 8th international workshop on Specification and verification of component-based systems (SAVCBS '09), ACM, New York, NY, USA, pp. 19-2, 2009.

[6] Barnett, M., et al., “The Spec# Springer-Verlag, Berlin, Heidelberg, Programming System: Challenges and Directions″ In Verified Software: Theories, Tools, Experiments, Lecture Notes In Computer Science, Vol.4171. pp. 144-152, 2005.

[7]Burdy, L., M. Huisman and M. Pavlova, “Preliminary design of BML: A behavioral interface specification language for Java bytecode″ Fundamental Approaches to Software Engineering, LNCS 4422, pp. 215-229, 2007.

[8]Barnett M, Chang B-YE, DeLine R, Jacobs B, Leino, “Boogie: a modular reusable verifier for object-oriented programs″ Formal Methods for Components and Objects, Revised Lectures, Vol 4111 of Lecture Notes in Computer Science. Springer, Heidelberg, pp 364 - 387, 2005.

[9] Claude Marché, “The Krakatoa Verification Tool for JAVA programs″ INRIA Team-Project Proval, 2 March 2011.

[10] Gary T. Leavens, Albert L. Baker and Clyde Ruby, “JML : a Java Modeling Language″ Department of Computer Science, 266 Atanasoff Hall Iowa State University, Ames, Iowa 50011-1040 USA, 18 September 1998.

[11] Mike Barnett and K. Rustan M. Leino, “Weakest-precondition of unstructured programs″ SIGSOFT Softw. Eng. Notes31, 82-87, 1 September 2005.

[12]Laurent Hubert, Nicolas Barré, Frédéric Besson, Delphine Demange, Thomas Jensen, Vincent Monfort, David Pichardie, and Tiphaine Turpin. Sawja: Static analysis workshop for java. Technical report, CNRS, INRIA, ENS Cachan, June 2010. Presented at the International Conference on Formal Verification of Object-Oriented Software (FoVeOOS) in June 2010.