자바 프로그램을 위한 뮤테이션 도구 구현

이효정[0] 마유승 김상운 권용래
한국과학기술원 전자전산학과
(hjlee[0],ysma,swkim,kwon)@salmosa.kaist.ac.kr

Implementation of a Mutation Tool for Java Programs

Hyo-Jeong Lee[0] Yu-Seung Ma Sang-Woon Kim Yong-Rae Kwon
Dept. of Electronic Engineering and Computer Science, KAIST

요 약

컴퓨터 성능의 향상으로 고비용의 수행을 요하는 뮤테이션 기법의 적용 가능성이 커지면서 뮤테이션 기법에 대한 연구가 다시 활성화되고 있다. 뮤테이션 기법에 대한 연구는 순차 프로그램에 대해서는 완성 단계인 것에 반해, 객체지향 프로그램에 대한 연구는 역사가 짧고 아직 초기 단계에 머무르고 있다.

본 논문에서는 현존하는 자바 뮤테이션 오퍼레이터를 모두 지원하는 뮤테이션 분석 도구인 MuJava/SC를 구현하고 이의 성능 개선 방안에 대해서 논의한다. MuJava/SC 구현에는 리플렉션 시스템을 사용하였는데 그 중에서 구조적 리플렉션이 객체지향 뮤테이션 도구의 구현에 적절히 이용됨을 보여주었다. MuJava/SC의 성능을 개선한다면 객체지향 프로그램의 뮤테이션 분석에 유용하게 사용될 수 있을 것이다.

1. 서 론

성공적인 소프트웨어 시험을 위해서는 효율적인 시험 데이터(test case)가 필요하다. 뮤테이션 분석(mutation analysis)[1]은 오류기반(fault-based) 기법을 바탕으로 시험 데이터의 효율성을 측정하기 위하여 개발된 방법이다. 하지만 뮤테이션 시험은 수행비용이 크므로 시험 수행 시 도구의 뒷받침이 절대적이다.

순차 프로그램을 위한 뮤테이션 기법에 대한 연구는 완성 단계인 것에 반해 객체지향 프로그램에 대한 연구는 그 역사가 길지않고 뮤테이션 오퍼레이터의 정의[3] 수준에 그치고 있다. 뮤테이션 오퍼레이터의 정의 후 이의 효율성을 입증하고 뮤테이션 시험의 실제 적용을 가능하게 해주기 위해서는 뮤테이션 분석을 지원해주는 도구가 우선적으로 필요하다. 현재까지 객체지향 프로그램을 위한 뮤테이션 분석 도구로는 Chevalley에 의해 개발된 도구[2]가 유일하게 알려져 있다. 하지만 이 도구는 현존하는 오퍼레이터들을 모두 지원해 주지 못하고 수행속도가 느리다.

본 논문에서는 Chevalley의 아이디어를 기반으로 현존하는 자바 뮤테이션 오퍼레이터를 모두 지원해주는 시험도구를 구현하여, 구조적 리플렉션(structural reflection)[7]이 객체지향 뮤테이션 도구에 적절히 사용될 수 있음을 보여준다. 그리고 뮤테이션 기법의 중요한 문제중의 하나인 수행 비용을 절약하는 방법을 제시한다.

본 논문의 구성은 다음과 같다. 2장에서는 객체지향 프로그램을 위한 뮤테이션 분석 기법에 대해 설명한다. 3장에서는 뮤테이션 도구 구현에 사용되는 구조적 리플렉션에 대해 설명한다. 4장에서는 우리가 구현한 뮤테이션 분석 도구인 MuJava/SC의 구현과 이의 성능향상 방법에 대해서 기술한다. 마지막으로 5장에서는 결론 및 향후 연구에 대해 기술한다.

2. 객체지향 프로그램을 위한 뮤테이션 분석

뮤테이션 분석은 주어진 프로그램 P에 대해서 오류를 삽입한 프로그램인 뮤턴트(mutant) P'을 구별할 수 있는 시험 데이터를 좋은 시험 데이터라 간주하여 시험 데이터의 효율성을 측정한다. 따라서 뮤테이션 분석을 위해서는 오류를 삽입하는 규칙을 정의하는 뮤테이션 오퍼레이터(mutation operator)들이 우선적으로 필요하며, 뮤테이션 분석은 이들을 적용시켜 뮤턴트들을 생성한 뒤 시험 데이터를 수행시킨 결과를 분석하는 과정으로 이루어진다. 이 때 막대한 수의 뮤턴트들이 생성,수행되므로 뮤테이션 분석 시 도구의 사용은 절대적이다.

객체지향 프로그램을 위한 뮤테이션 분석을 위해서는 객체지향 프로그램이 갖는 캡슐화(encapsulation), 상속(inheritance), 다형성(polymorphism) 등의 특성들을 고려한 뮤테이션 오퍼레이터가 필요하다. 하지만 프로그램 언어에 따라 객체지향 특성들이 약간씩 다른 형태로 지원되므로 각 프로그램 언어의 특성을 고려하여 뮤테이션 오퍼레이터가 정의 되어야 한다. 자바 언어에 대한 객체지향 특성을 고려한 뮤테이션 오퍼레이터[3]들은 체계적으로 정의되어 있으며 표1과 같다. 하지만 아직 이들의 효율성에 대한 실험결과는 나와있지 않다.

객체지향 프로그램을 위한 뮤테이션 분석도구로는 현재까지 Chevalley에 의해 개발된 도구[2]만이 발표되었는데 이 도구는 리플렉션[6,8]이 객체지향 뮤테이션 도구 구현에 적절히 적용됨을 보여주었다. 하지만 이 도구는 자바 뮤테이션 오퍼레이터들이 체계적으로 정의되기 전에 개발되었기 때문에 현존하는 모든 뮤테이션 오퍼레이터들을 모두 지원하고 있지 않고 성능 개선의 여지가 있다.

Operators	Description
AMC	Access modifier change
IHD	Hiding variable deletion
IHI	Hiding variable insertion
IOD	Overriding method deletion
IOP	Overridden method calling position change
IOR	Overridden method rename
ISK	*super* keyword deletion
IPC	Explicit call of a parent's constructor deletion
PNC	*new* method call with child class type
PMD	Instance variable declaration with parent class type
PPD	Parameter variable declaration with child class type
PRV	Reference assignment with other compatible type
OMR	Overloading method contents change
OMD	Overloading method deletion
OAO	Argument order change
OAN	Argument number change
JTD	*this* keyword deletion
JSC	*static* modifier change
JID	Member variable initialization deletion
JDC	Java-supported default constructor create
EOA	Reference assignment and content assignment replacement
EOC	Reference comparison and content comparison replacement
EAM	Accessor method change
EMM	Modifier method change

[표 1] 자바 뮤테이션 오퍼레이터

본 논문에서는 Chevalley의 기법을 바탕으로 표 1의 뮤테이션 오퍼레이터를 지원하는 뮤테이션 분석 도구를 구현하였으며, 나아가 이의 성능개선 방안에 대해서 논의한다

3. 리플렉션(Reflection)

리플렉션(reflection)[6,7,8]은 프로그램이 자기 자신을 검사하고 조작할 수 있도록 하고, 프로그램의 행위와 구현을 변경할 수 있게 해주는 기능이다. 따라서 리플렉션 시스템의 이용은 오류를 심으려는 클래스로부터 필요한 정보를 뽑아낸 뒤 클래스를 변경하여 오류를 심는 작업을 수행하는 객체지향 뮤테이션 도구 개발을 용이하게 해준다.

리플렉션은 변경시키려는 대상에 따라 행위적 리플렉션(behavioral reflection)과 구조적 리플렉션(structural reflection)으로 구분되나 이들은 완전히 독립적인 것은 아니다. 행위적 리플렉션은 메소드 호출이나 객체 생성 등의 제한된 행위의 변화만을 가능하게 해주며 구조적 리플렉션은 클래스나 메소드 같은 데이터의 구조 변경을 가능하게 해준다. 클래스 구조의 변화를 요하는 자바 뮤테이션 오퍼레이터를 적용하여 오류를 심기 위해서는 구조적 리플렉션이 적절하다. 따라서 본 장에서는 행위적 리플렉션보다 구조적 리플렉션에 좀더 초점을 맞추겠다.

구조적 리플렉션 방법은 변경이 일어나는 시점에 따라 아래의 두 가지로 나뉜다.

(1) 컴파일 시점 리플렉션(Compile-time Reflection)

컴파일 시점 리플렉션[8]은 프로그램의 변경이 컴파일 시에 이루어진다. 다시 말해, 소스코드를 변경한 다음 재컴파일 과정을 거쳐야 한다. 이 기법은 자바 컴파일러대신 해당 리플렉션 시스템을 위해 제작된 컴파일러를 사용해야 하며 대표적인 시스템으로는 OpenJava[3]가 있다.

(2) 적재 시점 리플렉션(Load-time Reflection)

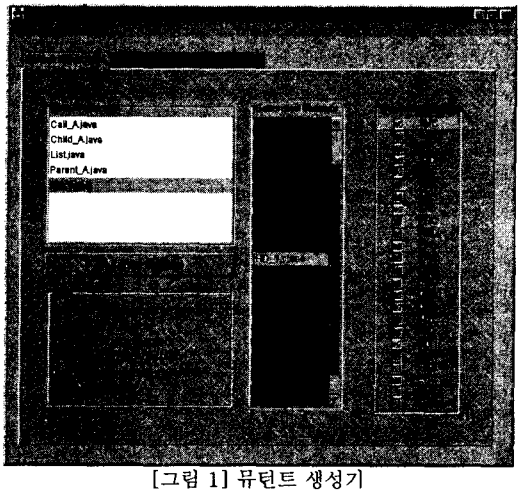
소스코드를 변화시키는 컴파일 시점 리플렉션 기법과는 달리 적재 시점 리플렉션은 소스코드의 컴파일 후 생성된 이진코드(byte code)를 직접 변환시킨다. 이 기법은 자바 컴파일러를 그대로 사용하며, 소스코드의 변경, 재컴파일 작업이 필요 없으므로 컴파일 시점 리플렉션보다 빠른 변경을 수행할 수 있다. 대표적인 시스템으로는 Javassist[7]가 있다.

4. MuJava/SC: 자바 뮤테이션 도구

본 논문에서는 현존하는 자바 뮤테이션 오퍼레이터를 모두 지원하는 뮤테이션 분석 도구 MuJava/SC를 구현하였다. MuJava/SC는 뮤턴트 생성부터 시험 데이터의 수행 및 결과 분석을 지원해 주며 크게 아래의 세 부분으로 구성된다.

(1) 뮤턴트 생성기

자바 프로그램에 사용자가 지정한 뮤테이션 오퍼레이터들을 적용시켜 뮤턴트들을 생성한다. 그림 1은 뮤턴트 생성기의 인터페이스를 보여준다.

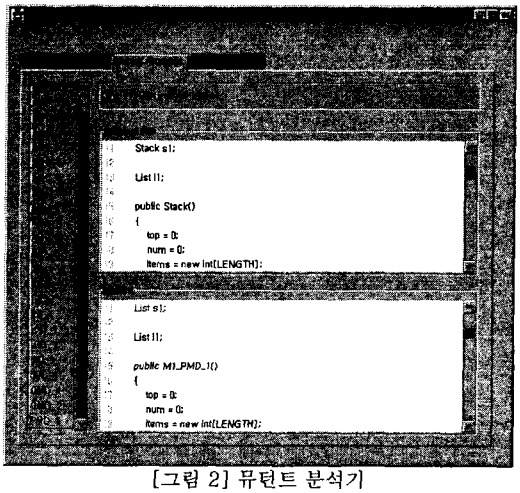


[그림 1] 뮤턴트 생성기

(2) 뮤턴트 분석기

생성된 뮤턴트를 분석하여 원 프로그램(original program)의 소스 코드의 어떤 위치에 어떤 오류가 심어졌는지 보여준다.

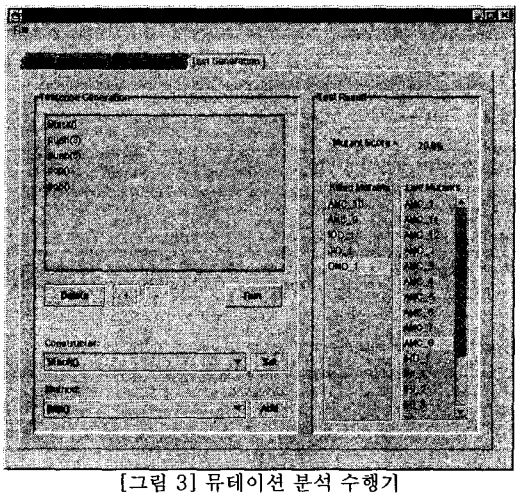
그림 2는 뮤턴트 분석기의 인터페이스를 보여준다. 왼쪽 부분에는 뮤턴트 생성기로부터 생성된 뮤턴트들이 나열되며, 이 중 보고자 하는 뮤턴트를 선택하면 오른쪽 부분에 원 프로그램과 뮤턴트의 소스코드의 차이점이 나타난다.



[그림 2] 뮤턴트 분석기

(3) 뮤테이션 분석 수행기

주어진 시험 데이터로부터 시험하려는 클래스에 대한 뮤테이션 분석을 수행한 뒤, 어떤 뮤턴트를 검출해 내는지를 보여준다. 뮤턴트의 검출정도는 뮤테이션 점수(mutation score)로 나타나며 뮤테이션 점수가 높을수록 효율적인 시험데이터로 여겨진다. 시험 데이터는 미리 만들어진 파일로부터 읽어 들이거나 사용자가 도구상에서 직접 작성할 수 있다. 그림 3은 뮤테이션 분석 수행기의 인터페이스를 보여준다. 인터페이스의 왼쪽 부분은 사용자가 시험 데이터를 쉽게 생성할 수 있도록 해주며 오른쪽 부분은 뮤테이션 분석 결과를 보여준다.



[그림 3] 뮤테이션 분석 수행기

MuJava/SC는 구조적 리플렉션 시스템을 이용하여 구현되었는데 이를 이용하여 멤버 변수(member variable)의 형 변환,삽입,삭제 등의 프로그램 변경을 필요로 하는 앞에서 언급된 모든 자바 뮤테이션 오퍼레이터들을 쉽게 구현할 수 있었다.

하지만 MuJava/SC은 수행 속도면에서 아직 비효율적이다. 특히 뮤턴트 생성부분이 전체 수행 시간의 90%를 차지할 정도였다. 따라서 도구의 성능 개선을 위해서는 뮤턴트 생성 시간을 줄이는 것이 우선적이다.

뮤턴트 생성시간이 많이 걸리는 원인으로는 간단하지만 성능이 좋지 않은 변경 알고리즘의 사용과 컴파일 시점 리플렉션 시스템인 OpenJava[8]의 이용을 들 수 있다. 앞 장에서 설명했듯이 컴파일 시점 리플렉션은 소스코드를 변경하여 재컴파일을 하여야 한다. 하지만 뮤턴트 생성은 소스 코드의 한 줄만을 변경하는 것이 대부분이다. 이처럼 극히 일부분의 변경을 위해 코드전체의 재컴파일을 수행하는 것은 매우 비효율적이다. 따라서 이 경우 이진 코드로부터 바로 변경을 수행하는 적재 시점 리플렉션이 컴파일 시점 리플렉션보다 적절해 보인다.

5. 결론 및 향후연구

본 논문에서는 구조적 리플렉션 시스템을 이용하여 자바 뮤테이션 오퍼레이션을 모두 지원하는 뮤테이션 분석 도구인 MuJava/CM을 구현하였으며 이의 성능 향상 방안에 대해 논의하였다. 리플렉션을 통해 오류를 심으려 하는 클래스로부터 필요한 정보를 쉽게 뽑아낼 수 있었으며, 특히 클래스나 메소드 같은 데이터의 구조 변경기능을 제공하는 구조적 리플렉션은 객체지향 프로그램의 뮤턴트 생성에 매우 효율적으로 이용됨을 보여주었다.

하지만 현재 개발된 도구는 그 수행 속도가 매우 느린 단점이 있다. 수행속도 개선을 위해서는 뮤테이션 스키마(mutation schema)[4]같은 빠른 알고리즘의 사용과 소스코드를 변경 시키는 컴파일 시점 리플렉션 시스템 대신 이진 코드를 직접 변경시키는 적재 시점 리플렉션 시스템을 사용하는 방법이 요구된다.

현재 본 연구팀은 MuJava/CM을 이용하여 제안된 자바 뮤테이션 오퍼레이터들의 효율성 검증 작업과, 적재 시점 리플렉션 시스템을 이용한 뮤테이션 도구의 구현 작업을 진행 중에 있다

Reference

[1] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, " Hints on test data selection: Help for the practing programmer," *IEEE Computer*, 11(4):34-41, April 1978

[2] P. Chevalley and P. Thevenod-Fosse, " A mutation analysis tool for Java programs," *Journal on Software Tools for Tecnology Transfer*, September 2001

[3] Y. S. Ma, Y. R. Kwon, and J. Offutt, " Inter-Class Mutation Operators for Java," to be presented at *ISSRE* 2002

[4] J. Offut and R. Untch, " Mutation 2000:Uniting the orthogonal," In *Proceedings of Mutation 2000: Mutation Testing in the Twentie th and the Twenty First Centuries*, pages 45-55, San Jose, CA, October 2000

[5] B. J. Choi, B. A. DeMillo, E. W. Krause, R. J. Martin, et al., " The Mothra tool set," *System Sciences*, 1989

[6] Sun Microsystems, " Reflection," http://java.sun.com/j2se/1.4/docs/guide/reflection/index.html, 2001

[7] S. Chiba, " Load-time Structural Reflection in Java," *ECOOP* 2000, pp.313-336, 2000

[8] M. Tatsubori, S. Chiba, et al.," OpenJava: A Class-based Macro System for Java," *LNCS 1826, Reflection and Software Engineering*, Springer-Verlag, pp.117-133, 2000

[9] S. Kim, J. Clark, and J. McDermid, " Class *mutation*: Mutation testing for object-oriented programs," *OOSS:Object-Oriented Software Systems*, October 2000