자바 컴파일러의 예외 메커니즘

조장우[U] 창병모
부산외국어대학교 컴퓨터전자공학과, 숙명여자대학교 전산학과
jjw@taejo.pufs.ac.kr, chang@cs.sookmyung.ac.kr

Exception Mechanism of Java Compiler

Jang-Wu Jo[U] Byeong-Mo Chang
Div. of Computer and Electronic Engineering, Pusan Univ. of Foreign Studies,
Dept. of Computer Science, Sookmyung Women's University

요 약

자바 컴파일러의 예외 처리 메커니즘을 기술한다. 자바 컴파일러 방식의 예외 상황 분석은 프로그래머의 선언에 의존하는 프로시져-내 분석이다. 본 논문에서는 자바 컴파일러 방식의 프로시져-내(intraprocedural) 분석과 선언에 의존하지 않는 프로시져-간(interprocedural) 예외 상황 분석기를 개발하고 실제 자바 프로그램에 적용시켜 봄으로서 자바 컴파일러의 예외 상황 분석의 문제점을 보인다.

1. 서론

견고한(robust) 소프트웨어 개발을 용이하게 하기 위해서 현대의 대부분의 프로그래밍 언어에서는 명시적인 예외 처리 메커니즘을 지원한다. 인터넷 프로그래밍 언어로서 세계적으로 많이 사용되는 자바 언어에서도 예외처리 메커니즘을 지원하고 있다.

자바의 예외는 JDK의 Exception 클래스와 이의 하위 클래스들로 제공되고 또한 프로그래머가 Exception 클래스의 하위클래스로 새로운 예외 클래스를 정의할 수 있다[1].

예외에 관련된 자바 구문으로는 예외를 발생시키는 **throw e** 문장이 있고 발생된 예외를 처리하는 **try ... catch( ) ...** 문장이 있다. 그리고 메소드 선언 부분에서 처리되지 않는 예외를 선언하는 **throws** 구문이 있다.

자바의 예외 메커니즘을 이용하여 프로그램의 안전에 허점이 발생하지 않기 위해서는, 프로그래머가 발생 가능한 예외 상황에 대한 적절한 예외처리기를 설치해야 한다. 그렇지 못한 경우 컴파일러가 미리 분석을 해서 실행되는 프로그램의 안전도를 증진시키는 것이 필요하다.

· 본 연구는 한국과학재단 목적기초연구 (2000-1-30300-009-2)지원으로 수행되었음.

본 연구에서는 집합-기반 분석(set-based analysis) 기법을 기반으로 프로그래머의 선언에 의존하는 자바 컴파일러 방식의 예외 상황 분석기와 프로그래머의 선언에 의존하지 않는 프로시져-간(interprocedural) 예외 상황 분석기를 개발함으로서 자바 컴파일러의 예외 상황 분석의 문제점을 보이고자 한다.

2. 연구 배경

자바는 메소드 내에서 처리되지 않는 예외들을 메소드의 프로토타입에 선언하도록 하고 있다. 이러한 조건은 프로그래머에 부담이 되어 광범위하게 선언하거나 실제 발생하지 않는 예외들을 선언할 수 있다. Sun 사의 JDK에서 제공하는 자바 컴파일러에서는 예외상황 분석을 통한 프로그램의 안전한 실행을 돕고 있다. 그러나 자바 컴파일러의 예외 상황 분석은 타입 정보와 프로그래머가 메소드 정의 시 선언한 예외 상황 정보에 전적으로 의존하는 프로시져-내(intraprocedural) 분석을 한다. 그러므로 프로그래머가 메소드의 프로토타입에 불필요한 예외들을 선언하거나 광범위하게 선언하는 경우 정확한 분석 결과를 내지 못하므로 정확한 예외처리를 하지 못하는 문제점이 있다. 이와 같은 상황의 간단한 예가 [그림1]이다.

```
class demo1{
  void demoProc1 ( ) throws Exception {
       try {
           demoProc2();
       } catch (Exception e) { ; }
  }

  void demoProc2 ( ) throws Exception {
       throw new IOException();
  }
```

그림 1.

3. 예외분석기의 설계

본 연구의 예외 상황 분석은 집합-기반 분석 기법을 기반으로 한다[4]. 예외 상황을 분석하기 위해서는 클래스 분석(class analysis) 정보가 필요하다. 클래스 분석이란 어떤 식이 가리키는 객체의 타입(클래스)를 정적으로 유추하는 분석이다[3]. 자바에 대한 집합-기반 클래스 분석은 [2]에 기술되어 있다. 본 연구는 클래스 분석 정보가 존재한다고 가정하고 예외 분석기를 설계한다.

수식 단위의 예외 상황 분석은 각각의 수식 e 에 대해 처리되지 않는 예외들을 가지는 집합 변수 Pe 를 정의해서 $Pe \supseteq$ *set-expression* 과 같은 집합-관계식(set-constraints)을 구성한다. 집합-관계식 $Pe \supseteq$ *set-expression*의 의미는 집합 변수 Pe 가 *set-expression* 이 나타내는 집합을 포함한다는 것이고 여러개의 집합 관계식들은 conjunctive 관계이다.

수식 단위의 예외 상황 분석은 이론적으로는 정확한 분석 결과를 제공하지만 집합 변수의 수가 수식의 수 만큼 필요하므로 분석 속도 면에서 실용적이지 못하다. 집합-기반 분석의 시간 복잡도는 집합 변수의 수가 N 이라고 할 때 N^3 이므로 실제적인 자바 프로그램의 분석에는 적절하지 않다. 그러므로 각각의 수식이 아닌 각 메소드와 try 블록에 집합 변수를 정의하여 집합 변수의 수를 줄임으로서 분석 속도면에서 실용적인 예외 상황 분석기를 설계한다. 메소드 수준의 시간 복잡도는 수식 수준과 같이 집합 변수의 수가 N 일 때 N^3 이지만 N 의 수를 줄이는 것이다.이와 같이 분석의 단위를 조절하는 기법은 SML 의 예외 상황 분석에 성공적으로 적용된 바 있다 [5]. 자바의 주요 구문에 대한 메소드 단위의 집합 기반 예외 상황 분석기의 설계는 [그림2]과 같다.

[그림2]의 **Class**(e_1)는 수식 e_1 에 대한 클래스 분석의 결과로 수식 e_1 이 가리키는 객체의 클래스를 나타낸다. $f\triangleright e$ 는 수식 e 가 메소드 f 내의 문장이란 의미이다.

[그림2]의 생성 규칙의 의미는 다음과 같다.

$f\triangleright$ throw e_1 $\Rightarrow P_f \supseteq$ **Class**(e_1)

throw e 가 메소드 f 의 구문일 때 메소드 f 에 대한

예외 상황 분석을 위한 도메인:
예외 클래스 이름들의 집합

집합 변수 P_f:
메소드 f에 대한 집합 변수로 f 에 대한 호출에서 발생되어 처리되지 않는 예외들을 위한 변수

집합 변수 P_g:
try 블록 g를 위한 집합 변수로 try 블록 g 내에서 발생되어 처리되지 않는 예외들을 위한 변수

집합 관계식의 형태:
$P_f \supseteq$ *set-expression*
$P_g \supseteq$ *set-expression*

집합-기반 분석:
각 P_f, P_g 에 대해 메소드 f, try 블록 g 내에서 발생되어 처리되지 않는 예외들을 계산한다.

생성 규칙 예:
$f\triangleright$ throw $e \Rightarrow P_f \supseteq$ **Class**(e)
$f\triangleright$ try e_0 catch (c_1 x_1) e_1
$\Rightarrow P_f \supseteq P_g - \{c_1\}^*$
$f\triangleright$ $e_0.m(e_1)$ $\Rightarrow$
1) intraprocedural(자바 컴파일러 방법)
$\{P_f \supseteq T_{c.m} \mid c \in$ **Class**(e_0), $m(x) \in c\}$

2) interprocedural
$\{P_f \supseteq P_{c.m} \mid c \in$ **Class**(e_0), $m(x) \in c\}$

여기서 $P_{c.m}$ 는 클래스 c 의 메소드 m 에 대한 집합변수이고 $T_{c.m}$ 는 클래스 c 의 메소드 m 에 대한 사용자 정의(throws specification)에 대한 집합 변수를 의미한다.

그림 2. 예외분석기의 설계

집합 변수 P_f 는 처리되지 않는 예외 **Class**(e_1)을 포함한다.

$f\triangleright$ try e_0 catch (c_1 x_1) e_1
$\Rightarrow P_f \supseteq P_g - \{c_1\}^*$

try 블록 e_0 에서 처리되지 않는 예외들은 집합 변수 P_g 가 가지고 있고 e_0 에서 발생한 예외들 중에서 catch 블록에서 처리될 수 있다. 그러므로 위 구문에서 처리되지 않은 예외들은 e_0 에서 발생한 예외들 중에서 클래스 c_1 의 하위클래스들을 제외한 예외들이다.

$f\triangleright$ $e_0.m(e_1)$ $\Rightarrow$
1) intraprocedural(자바 컴파일러 방법)
$\{P_f \supseteq T_{c.m} \mid c \in$ **Class**(e_0), $m(x) \in c\}$

2) interprocedural
$\{P_f \supseteq P_{c.m} \mid c \in$ **Class**(e_0), $m(x) \in c\}$

메소드 호출 $e_0.m(e_1)$ 에서 처리되지 않은 예외들은 호출된 메소드 $m(e_1)$ 에서 처리되지 않은 예외들이다. 자

바 컴파일러에서는 호출된 메소드에서 처리되지 않은 예외 $T_{c.m}$ 은 호출 된 메소드의 throws 구문에 선언된 예외들을 나타내고 프로시져-간 분석 방법에서는 호출된 메소드에서 처리되지 않은 예외 $P_{c.m}$ 은 실제 처리되지 않는 예외들을 나타낸다.

[그림2]의 생성 규칙을 이용하여 입력 자바 프로그램에 대한 집합-관계식을 구성하고 구성된 제약들의 해를 구하면 각 메소드와 try-catch 구문에서 처리되지 않은 예외들의 정보를 구할 수 있다.

4. 구현 및 실험

4.1 구현

3장에서 설계된 예외 상황 분석기는 2-패스로 구현된다. 첫 번째 패스에서는 입력 프로그램에 대해 집합-관계식을 구성한다. lex&yacc을 이용하여 예외상황 분석을 위한 집합-관계식 구성 규칙을 yacc 의 semantic action 형태로 기술한다. 두 번째 패스에서는 구성된 집합-관계식의 해를 계산한다. 집합-관계식의 해는 집합-관계식의 iterative-fixed point를 구하는 과정으로 계산된다. 이 과정은 프로그램에 존재하는 예외 클래스의 수가 유한하므로 반드시 종료하게 된다.

4.2 대상 프로그램

실험은 소스 코드가 공개되고 자주 사용되는 자바 응용 프로그램과 자바 애플릿을 대상으로 했다. 여기서 사용된 자바 프로그램들은 [6,7]에서 구할 수 있다. [표1]과 [표2]는 대상 프로그램에 대한 간략한 설명이다.

No.	Name	Description
1	JHLZIP	ZIP compressor
2	JHLUNZIP	ZIP uncompressor
3	com.ice.tar	Unix Tar Archive
4	Jess-Rete	Reasoning Engine of Jess

표 1

No.	Total classes	Total methods	Lines of Code
1	2	11	425
2	1	3	187
3	10	141	4025
4	1	98	1667

표 2

4.3 실험 결과

4.2절의 대상 프로그램에 대해 프로시져-내와 프로시져-간 예외 상황 분석을 적용한 결과는 [표3]과 같다. [표3]에서 unnecessary throws 는 발생하지 않는 예외들을 메소드의 프로토타입에 선언한 경우이고 broader throws 는 발생한 예외 클래스보다 상위의 클래스들을 선언한 경우이다. 이와 같은 경우 자바 컴파일러와 같이 프로시져-내 분석을 하면 불필요하고 광범위한 throws 구문을 검증하지 못해 적절한 예외처리가 어려운 경우가 발생하는 것을 알 수 있다.

No	unnessary and broader throws		uncaught exception	
	intraprocedural	interprocedural	intraprocedural	interprocedural
1	1	2	4	4
2	0	1	1	1
3	1	7	49	41
4	1	7	42	33

표 3

5. 결론

본 논문에서는 자바 컴파일러의 예외 메커니즘을 기술했다. 그리고 자바 컴파일러의 프로시져-내 예외 상황 분석기와 프로시져-간 예외 상황 분석기를 설계, 구현하고 대상 프로그램에 실험을 하였다. 자바 컴파일러의 프로시져-내 분석은 정확한 분석 결과를 내지 못하므로 정확한 예외 처리를 하지 못하는 문제점이 있음을 기술했다.

6. 참고문헌

[1] K.Arnold and J.Gosling, "The Java Programming Languages, Second Edition", Addison-Wesley, 1997

[2] B.Chang, K.Yi, and J.Jo, "Constraint-based analysis for Java", SSGRR2000 Computer and e-business Conference, August 2000, L'Aquila, Italy.

[3] G.Defouw, D.Grove, and C.Chambers, "Fast interprocedural class analysis", in Proceedings of 25th ACM SIGPLAN Symposium on Principles of Programming Languages, pages 222-236, Jan. 1998.

[4] N.Heintze, "Set-based program analysis", Ph.D thesis, Carnegie Mellon University, Oct, 1992.

[5] K.Yi and S.Ryu, "Towards a cost-effective estimation of uncaught exceptions in SML programs", In Lecture Note in Computer Science, volume 1302, pages 98-113, Springer-Verlag, Proceedings of 4th Static Analysis Symposium, Sep, 1997.

[6] http://www.jars.com

[7] http://www.gamelan.com