Java의 예외 제어 흐름을 포함한 제어 흐름 그래프 생성+*

조장우[0] 이정수
부산외국어대학교 컴퓨터공학과
jjw@taejo.pufs.ac.kr, jslee[0]@saejong.pufs.ac.kr

Constructing Control Flow Graph with Exceptional Control Flow for Java

Jang-Wu Jo[0] Jung-Su, Lee
Dept. of Computer Engineering, Pusan Univ. of Foreign Studies

요 약

제어 흐름 그래프는 프로그램의 문장들간의 제어 흐름 정보를 표현하는 방법이다. 제어 흐름 정보는 프로그램 분석과 테스팅 분야에서 필요로 하는 정보이다. 제어 흐름 정보가 정확할수록 정확한 분석 결과와 테스팅 결과를 구할 수 있다. 실제 자바 프로그램에서 예외 구문의 사용 빈도가 많으므로 예외 제어 흐름을 제어 흐름 정보에 포함해야 한다.

본 논문에서는 특정 분석에 무관하게 예외 제어 흐름을 포함하는 제어 흐름 그래프를 생성하는 일반적인 방법을 제안한다. 그리고 예외 제어 흐름을 포함하는 제어 흐름 그래프를 생성할 때, 정상 흐름과 예외 흐름을 분리해서 하는 방법을 제안한다.

1. 서 론

제어 흐름 그래프(Control Flow Graph)는 프로그램의 문장들간의 제어 흐름 정보를 표현하는 방법이다. 제어 흐름 그래프에서 노드는 문장을 나타내고 에지는 문장간의 제어의 흐름을 표현한다. 제어 흐름 정보는 프로그램 분석과 소프트웨어 공학 분야에서 필요한 정보이다. 예를 들어, 자료 흐름 분석(data flow analysis), 제어 종속 분석(control dependence analysis), 예외 분석(exception analysis), 리그레션 테스팅(regression testing), 프로그램 슬라이싱, 테스트 데이터 생성 등을 위해서 제어 흐름 정보가 필요하다. 제어 흐름 정보가 정확할수록, 정확한 분석 또는 테스팅 결과를 얻을 수 있다. 만일 틀린 제어 흐름 정보를 기반으로 분석 또는 테스팅을 수행하면, 틀린 결과를 얻을 수 있다.

자바 언어는 안전하고 강건한(robust) 프로그램 작성을 위해 예외 메커니즘을 지원한다. 그리고 Ryder[1]와 Sinha[2]의 연구에 따르면, 클래스들의 23.3%가 try 구문을 그리고 24.5%가 throw 구문을 사용하고, 메소드들 중의 16%가 예외 관련 구문들을 사용한다. 자바 프로그램에 대한 정확한 제어 흐름 정보를 위해서는 예외 흐름 정보를 포함해야 한다.

최근에 예외를 고려한 자바 프로그램에 대한 분석들이 진행되고 있다. Choi[3]는 자료 흐름 분석을 위해 예외 흐름 정보를 포함한 제어 흐름 그래프를 제안하였고, Ryder[4,5]는 예외 흐름을 고려한 points-to 분석과 자료 흐름 분석을 수행하였다. Sinha[2]는 예외를 고려한 제어 종속 분석과 슬라이싱을 위해 제어 흐름 그래프를 제안하였다.

그러나 기존의 방법들은 특정 분석에 종속적인 자료 흐름 그래프를 생성하고, 제어 흐름 정보 생성 시 정상 제어 흐름과 예외 제어 흐름을 동시에 생성한다. 본 논문에서는 특정 분석에 무관하게 예외 제어 흐름을 포함하는 제어 흐름 그래프를 생성하는 일반적인 방법을 제안한다. 그리고 예외 제어 흐름을 포함하는 제어 흐름 그래프를 생성할 때, 정상 흐름과 예외 흐름을 분리해서 하는 방법을 제안한다.

예외 제어 흐름 정보를 생성하기 위해서 집합기반 분석(set-based analysis)[6]을 이용한 예외 제어 흐름 분석을 설계한다. 예외 제어 흐름 분석의 결과는 순환 메소드(recursive method)로 인해 무한할 수 있으므로, 분석 결과를 유한하게 표현할 수 있는 방법을 제안한다. 그리고 예외 제어 흐름을 이용하여 예외를 포함한 제어 흐름 그래프를 생성한다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 정상 제어 흐름과 예외 제어 흐름간의 관계를 기술하고, 3장에서는 예외 제어 흐름 분석을 설계하고 해를 구한다. 4장에서는 예외 제어 흐름 정보를 이용하여 예외를 포함한 제어 흐름 그래프를 생성에 관해 기술하고, 5장에서는 결론을 기술한다.

2. 정상 제어 흐름과 예외 제어 흐름

제어 흐름 분석은 프로그램의 각 문장 *s*에 대해서, *s* 다음에 수행될 수 있는 문장들을 결정하는 것이다. 정상 제어 흐름은 기본적으로 프로그램의 쓰여진 순서대로 한 번씩 수행이 되고, 제어 구조를 바꾸는 구문은 조건문, 반복문, 그리고 메소드 호출이 있다.

예외 제어 흐름은 예외를 발생시키는 *throw* 구문에서

+ 본 연구는 한국과학재단 목적기초연구(2000-1-30300-009-2) 지원으로 수행되었음.
* 이 논문은 2002학년도 부산외국어대학교 학술연구조성비에 의해 연구되었음

시작한다. *throw* 문장 다음에 수행되는 문장은 발생된 예외를 처리하는 *catch* 블록이다. *throw* 문을 포함하는 *try-catch* 문장에서 발생된 예외의 *catch* 블록이 존재하면 해당 *catch* 블록이 수행되고, 존재하지 않으면 중첩된 *try-catch* 블록, 메소드 호출 관계의 역순으로 발생한 예외의 *catch* 블록을 찾아간다. *catch* 블록 다음에 수행되는 문장은 예외가 발생한 문장의 다음 문장이 아닌 해당 *try-catch* 문장의 다음 문장이다.

정상 제어 흐름과 예외 제어 흐름을 분리해서 구하기 위해서 두 흐름간의 관계를 규명하고자 한다. 이론적으로 두 흐름은 상호 의존적이다. 즉, 정확한 정상 제어 흐름을 구하기 위해서는 예외 흐름 정보를 필요로 하고, 역으로 정확한 예외 흐름을 구하기 위해서는 정상 제어 흐름 정보를 필요로 한다. 그러나 실제 자바 프로그램을 대상으로 두 흐름간의 상호 의존적인 경우의 발생 빈도를 실험한 [7]에서 상호 의존적인 경우가 아주 낮은 비율로 발생함을 알 수 있었다. 그러므로 두 흐름을 분리해서 구하더라도 정확도가 많이 감소되지 않음을 알 수 있다.

3. 예외 제어 흐름 분석

3.1 ExnJava

P	::=	C^*	program
C	::=	**class** c **ext** c {**var** x^* M^*}	class definition
M	::=	$m(x)$ = e [throws c^*]	method definition
e	::=	id	variable
	|	id := e	assignment
	|	**new** c	new object
	|	**this**	self object
	|	e ; e	sequence
	|	**if** e **then** e **else** e	branch
	|	**throw** e	exception raise
	|	**try** e **catch** (c x e)	exception handle
	|	$e.m(e)$	method call
id	::=	x	method parameter
	|	$id.x$	field variable
c			class name
m			method name
x			variable name

그림 1 ExnJava의 추상구문

본 연구에서 예외 제어 흐름 분석의 명료한 설명을 위해 예외 처리와 관련된 자바 언어의 구문들로 구성된 자바 언어의 부분 언어인 가상의 자바 언어 ExnJava를 정의하였다. 그림 1은 ExnJava에 대한 추상 구문(abstract syntax)이다.

3.2 예외 제어 흐름 분석 설계

본 연구에서는 예외 제어 흐름 분석을 집합-기반 분석으로 설계하였다. 집합-기반 분석은 구성 규칙(construction rule)을 이용해서 집합-관계식들(set constraints)을 구성하는 단계와 집합 관계식을 만족하는 해(solution)를 구하는 두 단계로 구성된다.

예외 제어 흐름을 분석하기 위해 본 논문에서는 *throw* 문, *try-catch*절, 메소드 선언과 같이 예외와 관련된 구조들의 레이블들만을 기록한다. 식 e가 l : e로 표기되는 레이블 l을 갖는다고 하자.

그림 2는 각 식에 대한 집합 관계식들을 나타낼 수 있는 규칙들이다. 프로그램의 모든 식 e는 집합 관계식 $X_e \supseteq se$를 갖는다. X_e는 식 e안에서 발생된 예외들의 제어 흐름을 모으기 위한 집합 변수이다. X_e의 아래첨자 e는 규칙이 적용되는 현재 식을 의미한다. 관계식 "$e \triangleright C$" 는 "집합 관계식 C가 식 e로부터 생성된다"고 읽는다.

3.3 분석의 해 구하기

집합 관계식의 해를 구하는 규칙 S를 설계한다. 그림 3은 해를 구하는 규칙들로 프로그램에서 가능한 자료 흐름 경로를 따라서 값들을 전달한다. 해를 구하는 규칙들 S를 적용함으로써 관계식 C의 무한 해 $lm_s(C)$를 계산할 수 있다. 이 해는 입력 프로그램에 순환 메소드 호출이 있을 수 있기 때문에 무한 해가 될 수 있다. 무한 해의 안전성은 프로그램의 모든 예외 제어 흐름이 해 $lm_s(C)$에 포함됨을 보임으로서 증명 가능하다.

$$\frac{X \supseteq X_1 \cup X_2}{X \supseteq X_1} \qquad \frac{X \supseteq X_1 \cup X_2}{X \supseteq X_2} \qquad \frac{X \supseteq Y \quad Y \supseteq \langle c^l, \tau \rangle}{X \supseteq \langle c^l, \tau \rangle}$$

$$\frac{X \supseteq X_1 \cdot l' \quad X_1 \supseteq \langle c, \tau \rangle}{X \supseteq \langle c^l, \tau \cdot, l' \rangle}$$

$$\frac{X \supseteq X_1 - \{c_1, \ldots, c_k\} \quad X_1 \supseteq \langle c^l, \tau \rangle \quad c \notin \{c_1, \ldots, c_k\}}{X \supseteq \langle c^l, \tau \rangle}$$

그림 3 해를 구하는 규칙

무한 해를 유한 해로 표현하는 것이 필요하다. 따라서 S 안에서 예외 전파 규칙을 수정해 유한 해를 구하기 위한 새로운 규칙을 설계한다. 기본 아이디어는 예외 제어 흐름을 이를 구성하는 에지(edge)들로 나타내는 것이다. 즉 예외 제어 흐름을 기록하기 위해서 예외 전파의 각 단계에서 발생된 예외와 에지를 구성하는 두 개의 레이블을 유지한다. 예외 제어 흐름을 위한 규칙을 아래와 같이 수정한다.

$$\frac{X \supseteq X_1 \cdot l' \quad X_1 \supseteq \langle c^l, \tau \rangle}{X \supseteq \langle c^l, \lfloor \tau \cdot l' \rfloor_2 \rangle}$$

여기서, $\lfloor l_1 \ldots l_n \rfloor_2 = l_{n-1} l_n, n \geq 2$

이 규칙은 발생된 예외의 식별자 c^l과 함께 마지막 두 개의 레이블을 기록함으로써 발생된 예외들의 전파를 시뮬레이트한다. 레이블들의 기록은 예외 전파의 모든 단계에서 수행되기 때문에 식별자 c^l과 함께 해에 포함된다. 유한 해의 안전성은 무한할 수 있는 해의 모든 경로들을 예외 전파 그래프 상에서 찾음으로써 보일 수 있다.

4. 제어 흐름 그래프 생성

예외 제어 흐름을 나타내기 위해 다음과 같이 해

$lm_s \cdot (C)$의 예외 제어 흐름 그래프를 정의한다.

정의 1. [예외 제어 흐름 그래프]
C는 프로그램 P를 위해 구성된 집합-관계식이라 하자. 해 $lm_s \cdot (C)$의 예외 제어 흐름 그래프는 그래프 $<V, E>$로 정의되면 V는 P 내의 레이블들의 집합이고 $E = \{l_1 \rightarrow^{c'} l_2 \mid \langle c', l_1 l_2 \rangle \in lm_s \cdot (C)(X),\ X$는 C 내의 집합 변수$\}$ 이다. $l_1 \rightarrow^{c'} l_2$는 c' 레이블을 갖는 l_1에서 l_2로의 에지를 말한다.□

레이블을 갖는 에지들을 따라가면서 유한 해에 대한 예외 제어 흐름 그래프는 쉽게 그릴 수 있다. 또한 예외를 포함하는 제어 흐름 그래프는 정상적인 제어 흐름 그래프에 예외 제어 흐름 그래프를 추가함으로서 생성할 수 있다.

분석에 따라서 다른 예외 제어 흐름 정보가 필요하면, 레이블을 기록하는 구문을 조절함으로서 필요한 예외 흐름 정보를 구할 수 있다. 예를 들어, 메소드 호출과 *try* 블록들과 같은 구문들에 레이블을 기록할 수 있다.

5. 결론

본 논문에서는 특정 분석에 무관하게 예외 제어 흐름을 포함하는 제어 흐름 그래프를 생성하는 일반적인 방법을 제안하였다. 그리고 예외 제어 흐름을 포함하는 제어 흐름 그래프를 생성할 때, 정상 흐름과 예외 흐름을 분리해서 생성하였다. 두 흐름 정보를 통합함으로서 제어흐름 그래프를 생성할 수 있음을 보였다. 분석에 따라서 다른 예외 제어 흐름 정보가 필요하면, 레이블을 기록하는 구문을 조절함으로서 필요한 예외 흐름 정보를 구할 수 있다. 또한 정상 흐름과 예외 흐름을 분리함으로서, 기존의 예외를 고려하지 않은 제어 흐름 분석 방법을 사용할 수 있다.

참고문헌

[1] B. G. Ryder, D. Smith, U. Kremer, M. Gordon, and N. Shah, "A static study of Java exceptions using JESP," Tech. Rep. DCS-TR-403, Rutgers University, Nov. 1999.

[2] S. Shinha and M. J. Harrold, "Analysis and Testing of Programs With Exception-Handling Constructs," IEEE Trans. on Software Engineering, Vol. 26, No. 9, 2000.

[3] J.-D. Choi, D. Grove, M. Hind, and V. Sarkar, "Efficient and precise modeling of exceptions for analysis of Java programs," in *Proceedings of PASTE '99 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, September 1999, pp. 21-31.

[4] R.K. Chatterjee, et al., "Complexity of concrete type-inference in the presence of exceptions," *Lecture Notes in Computer Science*, vol. 1381, pp. 57-74, Apr. 1998.

[5] R Chatterjee and B. G. Ryder, "Data-flow-based testing of object-oriented libraries," Tech. Rep. DCS-TR-382, Rutgers University, Mar. 1999.

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

[7] 이정수, 조장우, "클래스분석에서 예외분석의 필요성," 프로그래밍언어 논문지, 제 16권, 제1호, 2002.2

[New] $\text{new}\ \ c \triangleright \emptyset$

[FieldAss] $\dfrac{e_1 \triangleright C_1}{id.x := e_1 \triangleright \{X_e \supseteq X_{e_1}\} \cup C_1}$

[ParamAss] $\dfrac{e_1 \triangleright C_1}{x := e_1 \triangleright \{X_e \supseteq X_{e_1}\} \cup C_1}$

[Seq] $\dfrac{e_1 \triangleright C_1 \quad e_2 \triangleright C_2}{e_1 ; e_2 \triangleright \{X_e \supseteq X_{e_1} \cup X_{e_2}\} \cup C_1 \cup C_2}$

[Cond] $\dfrac{e_0 \triangleright C_0 \quad e_1 \triangleright C_1 \quad e_2 \triangleright C_2}{\text{if } e_0 \text{ then } e_1 \text{ else } e_2 \triangleright \{X_e \supseteq X_{e_0} \cup X_{e_1} \cup X_{e_2}\} \cup C_0 \cup C_1 \cup C_2}$

[FieldVar] $\dfrac{id \triangleright C_{id}}{id.x \triangleright C_{id}}$

[Throw] $\dfrac{e_1 \triangleright C_1}{l : \text{throw } e_1 \triangleright \{X_e \supseteq \langle c', l \rangle \cup X_{e_1}\} \cup C_1}\ c = class(e_1)$

[Try] $\dfrac{e_0 \triangleright C_0 \quad e_1 \triangleright C_1}{l' : \text{try } e_0 \text{ catch } (c_1\ x_1\ e_1) \triangleright \{X_e \supseteq ((X_{e_0} - \{c_1\}^*) \cup X_{e_1}) \cdot l'\} \cup C_0 \cup C_1}$

[MethCall] $\dfrac{e_1 \triangleright C_1 \quad e_2 \triangleright C_2}{e_1.m(e_2) \triangleright \{X_e \supseteq X_{c.m} \mid c \in \text{Class}(e_1), m(x) = e_m \in c\} \cup \{X_e \supseteq X_{e_1} \cup X_{e_2}\} \cup C_1 \cup C_2}$

[MethDef] $\dfrac{e_m \triangleright C}{l' : m(x) = e_m \triangleright \{X_{c.m} \supseteq X_{e_m} \cdot l'\} \cup C}\ m \in c$

[ClassDef] $\dfrac{m_i \triangleright C_i \quad i = 1, \ldots, n}{\text{class } c = \{\text{var } x_1, \ldots, x_k, m_1, \ldots, m_n\} \triangleright C_1 \cup \ldots \cup C_n}$

[Program] $\dfrac{c_i \triangleright C_i \quad i = 1, \ldots, n}{c_1, \ldots, c_n \triangleright C_1 \cup \ldots \cup C_n}$

그림 2 예외 제어 흐름 분석을 위한 구성 규칙