정적분석을 이용한 자바 프로그램의 예외 전파 시각화

(Visualization of Exception Propagation for Java Programs based on Static Analysis)

허 순 희 † 창 병 모 ††
(Soonhee Her) (Byeong-Mo Chang)

요 약 본 논문에서는 자바 프로그램의 예외 전파 경로를 추정하기 위한 집합 기반 분석을 제시하고 분석 결과를 이용하여 예외 전파 경로를 시각적으로 보여주는 시각화기를 제시한다. 본 논문에서는 예외 전파 분석기 및 시각화기를 구현하였다. 프로그래머는 시각화기를 이용하여 예외 전파 경로를 따라가면서 처리되지 않는 예외들을 감지하고 예외들을 보다 효과적으로 처리할 수 있다.

키워드 : 자바, 예외 전파, 예외 분석, 집합 기반 분석

Abstract This paper presents a static analysis based on set-based framework which estimates exception propagation paths of Java programs, and a visualization tool which visualizes propagation paths of exceptions using the static analysis information. We have implemented the exception propagation analysis and a visualization tool, which can guide programmers to handle exceptions more effectively

Key words : Java, exception propagation, exception analysis, set-based analysis

1. 서 론

자바에서의 예외는 프로그래머가 정의하고 발생시키고 처리할 수 있다. 프로그램의 안전에 허점이 발생하지 않도록 하기 위해서 프로그래머는 예외 처리 메커니즘을 이용하여 발생 가능한 예외에 대한 적절한 예외 처리기를 적절한 위치에 설치해야 한다. 그렇지 못한 경우에는 처리되지 않은 예외들이 프로그램의 실행을 멈출 수 있기 때문에 컴파일 시에 이를 확인하는 것이 매우 중요하다. 처리되지 않는 예외들을 판단하는 예외 상황 분석들은 지금까지 많이 연구되어 왔다[1,2,3].

지금까지의 연구들은 처리되지 않는 예외 정보만을 제공할 뿐 예외가 발생되어 전파되는 경로에 대한 정보는 제공하지 않고 있다. 예외 전파 정보는 프로시저-간 제어 흐름 그래프[4]를 생성하고, 예외 전파를 시각화하고, 프로그램에서 예외와 관련된 부분을 잘라내기 위해 필요한 정보이다.

본 논문에서는 자바 프로그램의 예외 전파 경로를 추정하기 위한 집합 기반 분석을 제안하고 분석 결과를 이용하여 예외 전파 경로를 시각적으로 보여주는 시각화기를 제시한다. 예외 전파 분석기는 자바 컴파일러 전단부인 Barat[5]을 기반으로 하여 구현하였으며 예외전파경로를 시각화하는 시각화기를 공개 소스 개발 환경인 Jipe[6] 상에 구현하였다. 시각화기에서는 예외 전파 분석 정보를 이용하여 예외전파그래프를 구성하고 이를 이용하여 선택된 예외의 전파 경로를 시각적으로 보여준다. 프로그래머는 시각화기를 이용하여 예외 전파 경로를 따라가면서 처리되지 않는 예외들을 감지하고 예외들을 보다 효과적으로 처리할 수 있다.

본 논문의 구성은 다음과 같다. 2장은 자바 언어에서의 예외에 대해서 설명하고 3장에서는 예외전파분석에 대해서 설명한다. 4장에서는 구현에 대해서 설명하고 5장에서 결론을 맺는다.

· 본 연구는 숙명여자대학교 2002년도 교내연구비 지원에 의해 수행되었음
† 비 회 원 : 숙명여자대학교 전산학과
hsh@cs.sookmyung.ac.kr
†† 종신회원 : 숙명여자대학교 전산학과 교수
chang@cs.sookmyung.ac.kr
논문접수 : 2002년 8월 8일
심사완료 : 2003년 3월 19일

2. 자바 언어에서의 예외

본 논문에서는 정적 분석을 간단하게 표현하기 위해서 자바 예외 구조들을 포함한 가상의 자바 언어를 고려한다. 그림 1은 가상 자바 언어를 위한 추상 구문이다.

자바에서 예외는 *throw* 문에 의해 발생될 수 있다. *try* e_0 *catch* (c x e_1) 문은 e_0 내에서 발생된 예외를 포착하여 처리하는 문장으로 c, x, e_1는 각각 처리할 예외 클래스 이름, 포착된 예외가 바인딩되는 변수, 그리고 예외 처리 코드를 의미한다. 만약 발생된 예외가 클래스 c에 해당되지 않으면 발생된 예외는 다른 처리기를 만날 때까지 호출한 메소드들의 호출 체인을 따라 역으로 전파된다. 프로그래머는 메소드 정의 시 메소드 내에서 발생되었으나 처리되지 않는 예외를 메소드 머리부에 선언해야 한다.

[그림 2]는 예외 전파를 보여주는 간단한 예제 프로그램이다.

P	::= C^*	프로그램
C	::= class c ext c { varx^* M^* }	클래스 정의
M	::= $m(x)$ = e[throws c^*]	메소드 정의
e	::= id	변수
	| id : = e	배정
	| new c	객체 생성
	| *this*	객체 자신
	| e ; e	순차
	| if e then e else e	분기
	| throw e	예외 발생
	| try e catch (c x e)	예외 처리
	| $e.m(e)$	메소드 호출
id	::= x	메소드 매개변수
	| $id.x$: = e	필드 변수
c		클래스 이름
m		메소드 이름
x		변수 이름

그림 1 자바 언어의 추상 구문

```
class Demo{
  public static void main(String[] args ) throws E2 {
     try {
        m1( );
     } catch (E1 x) { ; }
     ...
     m3( );
  }

  void m1( ) throws E1 {
     m2( );
  }

  void m2( ) throws E1 {
     throw new E1();
  }

  void m3( ) throws E2 {
     if (...) throw new E2();
     if (...) m3( );
  }
}
```

그림 2 예외전파를 위한 예제 프로그램

메소드 *m2*에서 발생된 예외 *E1*이 메소드 *m2*와 *m1*을 통과해서 전파되고 *main* 메소드의 *try*−*catch* 블록에서 예외가 처리된다. 예외 *E2*는 메소드 *m3*으로부터 발생된다. 만약 예외 *E2*가 전달되면 *main* 메소드까지 전파되고 처리되지 않는다. 메소드 *m3*는 또한 재귀 호출을 갖기 때문에 전달된 예외 *E2*가 재귀 호출을 통해 *m3*에 역으로 전달될 수 있다.

3. 예외 전파 분석

예외전파분석은 집합-기반 분석 틀을 기반으로 설계하였다[7]. 집합-기반 분석은 집합-관계식들을 구성하고 해를 구하는 두 개의 부분으로 구성된다. 이 장에서는 우선 집합-관계식의 표기법을 기술하고, 입력 프로그램의 모든 식에 대해서 발생될 예외들의 경로를 추정하는 집합-관계식 시스템을 보일 것이다.

3.1 집합 관계식

X가 집합 변수(set variable)이고 se가 집합 식(set expression)일 때, 각 집합 관계식은 $X \supseteq se$의 형태를 갖는다. 집합 관계식 $X \supseteq se$의 의미는 집합 X가 집합식 se로 표현된 집합을 포함한다는 것이다. 집합 관계식은 아래와 같은 형태를 갖는다.

$se \rightarrow \langle c^l, l \rangle$	l로부터 발생된 예외
| X	집합 변수
| $se \bigcup se$	합집합
| $se - \{c_1, \cdots, c_n\}$	예외 처리
| $se \cdot l$	예외 전파

throw 식으로부터 발생된 예외는 $\langle c^l, l \rangle$로 나타낸다. 여기서 c는 예외의 이름 즉 *throw* 식의 위치 즉 레이블이다. 예외의 발생지점을 명시하기 위하여 발생된 예외의 이름과 함께 *throw* 식의 레이블이 사용된다. c^l을 발생된 예외의 식별자라 한다.

집합 식 $se-\{c_1,\cdots,c_n\}$는 예외들을 처리하기 위한 식이다. 집합 식 $se \cdot l$은 se에 레이블 l을 붙여 예외 전파 경로를 기록한다. 위의 집합 관계식들은 모든 연산자가 단조 증가하고, 각 관계식의 왼쪽이 하나의 변수이기 때문에 프로그램의 최소 솔루션이 존재한다[7]. 여기서 관계식들의 모임 C의 최소 해를 $lm(C)$라고 쓴다.

3.2 집합 관계식 구성

예외 전파 경로를 추적하기 위해 본 논문에서는 *throw*문, *try*−*catch*절, 메소드 선언과 같이 예외와 관련된 구조들의 레이블들만을 기록한다. 식 e가 $l:e$로 표기되는 레이블 l을 갖는다고 하자. 만약 좀 더 자세한 경로 정보가 필요하면, 메소드 호출과 *try* 블록들과 같은 다른 식에 레이블들도 기록할 수 있다.

그림 3은 각 식에 대한 집합 관계식들을 나타낼 수 있는 규칙들이다. 프로그램의 모든 식 e는 집합 관계식 $X_e \supseteq se$를 갖는다. X_e는 식 e 안에서 발생된 예외들의 전파 경로를 모으기 위한 집합 변수이다. X_e의 아래첨자 e는 규칙이 적용되는 현재 식을 의미한다. 관계식 "$e \rhd C$"는 "집합 관계식 C가 식 e로부터 생성된다"고 읽는다.

예외 분석을 위해서 각 식의 클래스 정보가 필요하다. 본 논문에서는 각 식의 타입을 이용하였다[8,9]. [그림 3]에서 레이블 l을 가진 *throw* 식에 대한 규칙을 살펴보자. $c = class(e_1)$이 예외의 이름 즉 예외 클래스이고, l이 예외의 발생지점인 *throw* 문장의 레이블일 때 $\langle c^l, l \rangle$는 발생된 예외 e_1을 나타낸다. c^l은 발생한 예외의 식별자로 사용된다. 발생 전에 e_1 내부에서 처리되지 않는 예외가 있을 수 있으므로 X_{e_1}를 포함한다.

그림 3에서 레이블 l'을 가진 *try* 식에 대한 규칙을 살펴보자. e_0로부터 발생된 예외가 클래스 c_1에 의해 포착되면 발생된 예외는 x_1에 바인딩된다. 예외를 포착한 후에, e_1 내에서도 예외가 발생될 수 있다. 이 식에서 처리되지 않는 예외들은 예외 전파 경로를 기록하기 위한 레이블 l'을 첨부한다. 따라서 $\{c\}^*$가 자신을 포함하는 클래스 c의 모든 하위 클래스들을 나타낼 때, $X_e \supseteq ((X_{e_0} - \{c_1\}^*) \cup X_{e_1}) \cdot l'$이다.

그림 3에서 메소드 호출에 대한 구성 규칙을 살펴보자. 호출 식에서의 처리되지 않은 예외들은 우선 부분식 e_1, e_2로부터의 예외들을 포함한다: $X_e \supseteq X_{e_1} \cup X_{e_2}$. 메소드 $m(x) = e_m$은 클래스 $c \in classes(e_1)$ 안에 정의된 것이다. 따라서, 처리되지 않은 예외들에 대해 $X_e \supseteq X_{c.m}$이다(아래첨자 $c.m$은 클래스 c의 메소드 m에 대한 인덱스이다).

그림 3에서 레이블 l'을 가진 메소드 정의에 대한 규칙을 살펴보자. 이 메소드 m으로부터의 처리되지 않은 예외들은 메소드 몸체 e_m으로부터의 예외들을 포함하는데 예외 전파 경로를 기록하기 위한 레이블 l'을 첨부한다.

그림 2 프로그램에 집합 관계식 구성 규칙을 적용함으로써 그림 4와 같은 집합 관계식을 구성할 수 있다. 집합-관계식들을 이용할 때 이해를 돕기 위해 레이블들 대신에 단순화된 문장을 사용했다.

[*New*] $\text{new } c \rhd \emptyset$

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

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

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

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

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

[*Throw*] $\dfrac{e_1 \rhd C_1}{l : \text{ throw } e_1 \rhd \{\langle c^l, l \rangle \cup X_{e_1}\} \cup C_1} \quad c = class(e_1)$

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

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

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

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

[*Program*] $\dfrac{C_i \rhd C_i, \ i = 1, \cdots, n}{C_1, \cdots, C_n \rhd C_1 \cup \cdots \cup C_n}$

그림 3 집합 관계식 구성 규칙

$$
\begin{array}{ll}
X_{main} & \supseteq X_{try\ \ catch} \cdot main \\
X_{main} & \supseteq X_{m3} \cdot main \\
X_{try\ \ catch} & \supseteq (X_{m1} - \{Exception\}^*) \cdot try - catch \\
X_{m1} & \supseteq X_{m2} \cdot m1 \\
X_{m2} & \supseteq X_{throwE1} \cdot m2 \\
X_{throwE1} & \supseteq \langle E1, throwE1 \rangle \\
X_{m3} & \supseteq X_{throwE2} \cdot m3 \\
X_{throwE2} & \supseteq \langle E2, throwE2 \rangle \\
X_{m3} & \supseteq X_{m3} \cdot m3
\end{array}
$$

그림 4 집합 관계식

3.3 집합 관계식의 해 구하기

우선 집합 관계식 해를 구하는 규칙 S를 설계한다. 해를 구하는 규칙들 S를 적용함으로써 관계식 C의 무한 솔루션 $lm_s(C)$를 계산할 수 있다. 이 해는 입력 프로그램에 재귀 호출이 있을 수 있기 때문에 무한 해가 될 수 있다.

그림 5에 있는 규칙들은 프로그램에서 가능한 데이타 흐름 경로를 따라서 값들을 전파한다. 각 전파 규칙은 복잡한 집합 관계식을 보다 간단한 집합 관계식으로 나누며, 식들간의 값들의 흐름을 시뮬레이트하게 된다.

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

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

$$
\frac{X \supseteq X_1 - \{c_1, \cdots, c_k\} \quad X_1 \supseteq \langle c', \tau \rangle \quad c \notin \{c_1, \cdots, c_k\}}{X \supseteq \langle c', \tau \rangle}
$$

그림 5 집합 관계식 해 구하는 규칙 S

예외 전파 경로를 위한 규칙을 살펴 보자:

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

이 규칙은 X_1의 예외 경로 τ에 레이블 l'을 첨부함으로써 발생된 예외들의 전파 경로를 시뮬레이트한다. 다른 규칙들은 집합 식에 부합하는 의미를 따라 진행된다.

그림 3의 규칙 S를 적용함으로써 집합 관계식 C의 해 $lm_s(C)$를 구할 수 있다. 해의 안전성은 아래와 같이 기술할 수 있다.

정리 1. P는 입력 프로그램이고 C는 [그림 3]의 규칙들에 의해 구성된 집합 관계식이라 하자. P의 모든 예외 전파 경로는 해 $lm_s(C)$에 포함된다.

그림 4의 집합 관계식 C에 대한 무한 해는 규칙 S를 적용해서 구할 수 있다. 다음은 해의 일부분이다. 이 해는 처리되지 않는 예외들을 포함한 메소드가 재귀 호출되는 경우에 무한 해를 갖게 된다.

$$
\begin{array}{ll}
lm_s(C)(X_{m1}) & \supseteq \{\langle E1, throwE1 \cdot m2 \cdot m1 \rangle\} \\
lm_s(C)(X_{m2}) & \supseteq \{\langle E1, throwE1 \cdot m2 \rangle\} \\
lm_s(C)(X_{m3}) & \supseteq \{\langle E2, throwE2 \cdot m3 \rangle, \\
 & \quad \langle E2, throwE2 \cdot m3 \cdot m3 \rangle, \\
 & \quad \langle E2, throwE2 \cdot m3 \cdot m3 \cdot m3 \rangle, \cdots\} \\
lm_s(C)(X_{main}) & \supseteq \{\langle E2, throwE2 \cdot m3 \cdot main \rangle, \\
 & \quad \langle E2, throwE2 \cdot m3 \cdot m3 \cdot main \rangle, \\
 & \quad \langle E2, throwE2 \cdot m3 \cdot m3 \cdot m3 \cdot main \rangle, \cdots\}
\end{array}
$$

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

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

여기서,

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

이 규칙은 발생된 예외의 식별자 c'과 함께 마지막 두 개의 레이블을 기록함으로써 발생된 예외의 전파를 기록한다. 레이블들의 기록은 예외 전파의 모든 단계에서 수행되기 때문에 식별자 c'과 함께 해에 포함한다.

S'을 S의 전파 규칙을 새로운 규칙으로 대체한다. 새로운 해를 구하는 규칙 S'을 적용함으로써 집합 관계식 C의 최소 해 $lm_s'(C)$를 구할 수 있다. 본 연구에서는 예외 이름과 레이블의 수가 유한하기 때문에 S'의 해는 유한하다.

새로운 규칙 S'을 적용함으로써 그림 4의 집합 관계식 C에 대한 해를 다음과 같이 구할 수 있다.

$$
\begin{array}{ll}
lm_s(C)(X_{m1}) & \supseteq \{\langle E1, m2 \cdot m1 \rangle\} \\
lm_s(C)(X_{m2}) & \supseteq \{\langle E1, throwE1 \cdot m2 \rangle\} \{\langle E2, m3 \cdot m3 \rangle\} \\
lm_s(C)(X_{m3}) & \supseteq \{\langle E1, throwE2 \cdot m3 \rangle, \langle E2, m3 \cdot m3 \rangle\} \\
lm_s(C)(X_{main}) & \supseteq \{\langle E2, m3 \cdot main \rangle\}
\end{array}
$$

예외 전파 경로를 나타내기 위해 다음과 같이 해 $lm_s(C)$의 예외 전파 그래프를 정의한다.

정의 1. C는 프로그램 P를 위해 구성된 집합-관계식이라 하자. 해 $lm_s(C)$의 예외 전파 그래프는 그래프 $\langle V, E \rangle$로 정의되며 V는 P 내의 레이블들의 집합이고 $E = \{l_1 \rightarrow^{c'} l_2 \mid \langle c', l_1 l_2 \rangle \in lm_s(C)(X),\ X$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 예제 프로그램의 유한 해에 대한 예외 전파 그래프를 다음과 같이 쉽게

그릴 수 있다.

$$throwE1 \dashrightarrow^{E1} m2 \quad m2 \rightarrow^{E1} m1 \quad throwE2$$
$$\dashrightarrow^{E2} m3 \quad m3 \rightarrow^{E2} main \quad m3 \dashrightarrow^{E2} m3$$

유한 해의 안전성은 무한할 수 있는 해의 모든 경로들을 예외 전파 그래프 상에서 찾음으로써 보일 수 있다.

정리 2. $lm_S(C)$와 $lm_S(C)$는 각각 해를 구하는 규칙 S와 S'을 적용해서 구한 집합 관계식 C의 해라고 하자. $lm_S(C)$ 내의 모든 예외 전파 경로 $\langle c', \tau \rangle$에 대해 $lm_S(C)$의 예외 전파 그래프에 c'로 레이블된 경로 τ가 존재한다.

4. 구 현

이 장에서는 아래의 내용을 고려해서 예외 전파 분석을 구현하고 이 분석 정보를 이용해서 시각화 시스템을 구현한다. 이 장에서 구현할 시스템은 그림 6과 같이 입력된 자바 프로그램에 대해 집합 관계식들을 구성하고, 집합 관계식들의 해를 구하고, 해에 대한 예외 전파 그래프를 구성하고, 예외 전파 경로를 구성하고, 이를 시각화하는 5개의 하위 시스템으로 구성된다.

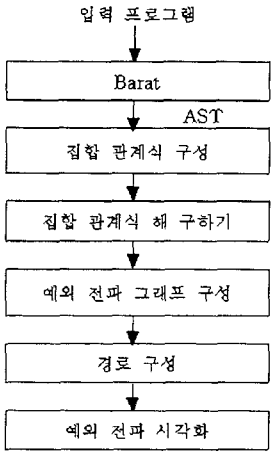

그림 6 시스템 구조

본 논문은 Java 언어로 구현된 두 가지 시스템, Barat[5]과 Jipe[6]를 기반으로 구현하였다. 예외전파 분석은 Java 컴파일러 전단부인 Barat을 기반으로 하여 구현하였으며 예외전파 시각화는 공개 소스 통합 개발 환경인 Jipe위에 구현하였다.

먼저 Barat은 Java 프로그램을 입력으로 받아 AST를 구성하고 타입 분석을 통해 타입 정보를 제공해준다. 또한 구성된 AST를 횡단할 수 있는 프로그래머 인터페이스를 제공한다. 본 연구에서는 Barat를 이용하여 이미 구현된 예외 분석기인 ExnAnal[1]를 확장하여 예외 전파 분석을 구현하였다. 예외 전파 분석은 두 개의 패스로 구현된다. 첫 번째 패스는 입력된 자바 프로그램에 대해서 AST를 횡단하면서 그림 9의 집합 관계식 구성 규칙을 적용하여 집합 관계식을 구성한다. 두 번째 패스에서는 구성된 집합-관계식들의 해를 집합-관계식의 고정점을 구하는 과정으로 계산한다.

다음 단계에서는 예외 전파 분석의 해를 이용하여 예외 전파 그래프를 구성한다. 이 그래프는 예외 전파 경로를 기록하기 위한 에지들로 구성되며 에지는 두 개의 레이블을 표현된다. 발생된 예외 c'가 l_1에서 l_2로 전파되는 경우 $l_1 \dashrightarrow^{c'} l_2$처럼 레이블을 갖는 에지 형태로 표현된다.

다음 단계에서는 사용자가 특정 예외를 선택하는 경우에 선택된 예외에 대해서 예외 전파 그래프로부터 그 예외의 전파 경로를 구성한다. 이 구성된 예외 전파 경로를 이용하여 특정 예외의 전파 경로를 사용자의 선택에 따라 한 단계씩 시각적으로 보여준다.

이 시각화 시스템은 공개 소스 통합 개발 환경인 Jipe 시스템[6] 위에 구현하였다. 본 연구에서는 Jipe 시스템 위에 하나의 새로운 메뉴(ExceptionBrowser) 형태로 예외 전파 시각화 도구를 추가하였다. 이 시각화 도구는 메소드, 예외 리스트, 예외전파 경로를 위한 세 개의 하위 창을 갖는 하나의 창 형태로 구현하였으며 예외 전파 분석 결과를 이용하여 사용자의 선택에 따라 선택된 예외의 전파 경로를 한 단계씩 시각적으로 보여준다.

ExceptionBrowser 실행 과정을 예를 들어서 살펴보도록 하자. Jipe 시스템 상에서 그림 7에서처럼 메뉴에서 ExceptionBrowser를 선택한다. 그림 7은 예제 프로그램도 보여주고 있다.

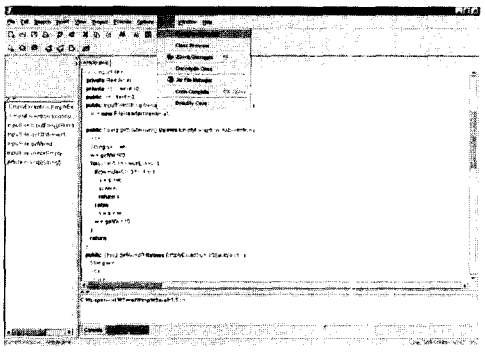

그림 7 예제 프로그램 및 Excetpion Browser 시작

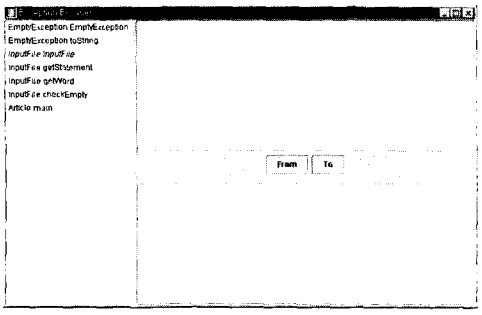

그림 8 ExceptionBrowser의 초기화면

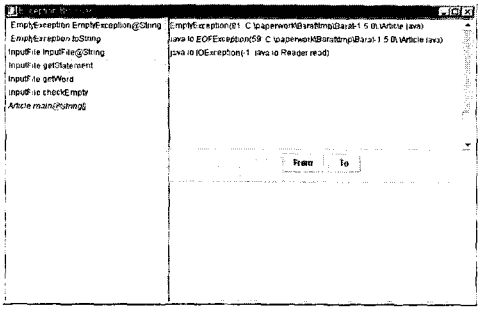

그림 9 예외 리스트 보여주기

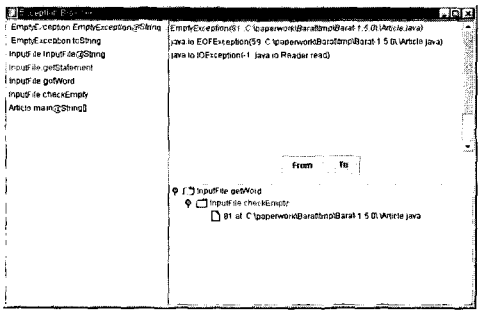

그림 10 예외가 전파되어 오는 경로

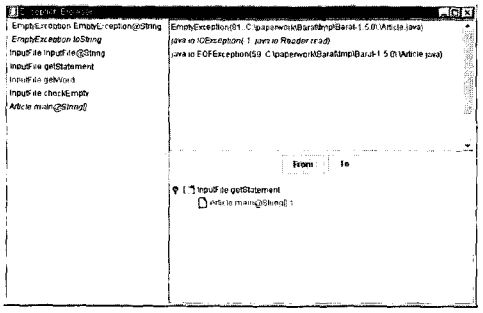

그림 11 예외가 전파되어 나갈 경로

그림 8은 ExceptionBrowser의 초기 화면이다. 초기 화면에서는 우선 화면의 왼쪽에 Jipe 에디터에 열려있는 자바 소스 파일 내의 메소드 리스트를 *클래스이름 · 메소드이름* 형태로 보여준다. 화면의 우측 상단은 *클래스이름 · 메소드이름* 리스트에서 선택한 메소드에 대해 그 메소드에서 발생할 수 있는 처리되지 않는 예외들의 리스트를 보여주는 부분이고, 우측 하단은 처리되지 않는 예외들 중 선택된 예외에 대해 그 예외의 전파 경로를 보여주는 부분이다. 마지막으로 두 개의 버튼 From과 To가 있다. From 버튼을 클릭하면 현재 선택된 예외가 어디에서부터 전파되어 왔는지를 보여주고, To 버튼은 선택된 예외가 어디로 전파되어 나갈지의 경로를 보여준다.

왼쪽 화면에 나타난 메소드 리스트에서 처리되지 않는 예외 정보를 보고자 한다면 리스트에서 메소드를 선택한다. 찾은 예외들의 리스트는 그림 9와 같이 화면의 우측 상단에 보여준다.

오른쪽 상단에 처리되지 않는 예외의 리스트가 나오면 From 버튼을 누르고 예외 리스트 중의 예외를 하나 선택하면 선택한 메소드까지 예외가 전파되어 오는 경로를 그림 10과 같이 보여준다. 또한 To 버튼을 누르고 예외 리스트의 예외를 선택하면 선택한 메소드로부터 전파되어 나갈 경로를 그림 11과 같이 보여준다.

본 연구에서 구현한 시스템의 의미를 정리하면 다음과 같다.

(1) 지금까지 예외 분석들은 처리되지 않는 예외에 대한 정보는 제공할 수 있으나 그 예외의 전파 경로에 대한 정보는 제공하지 못하였다[1-4]. 본 연구는 처리되지 않는 예외 정보뿐만 아니라 각 예외의 전파 경로 정보를 제공한다는 점에서 의의가 있다.

(2) 지금까지 예외 분석 정보는 단지 텍스트 형태로 제공되거나 컴파일러 최적화 등에 이용되었다[1-4]. 본 연구에서는 예외 전파를 포함한 예외 분석 정보를 사용자에게 시각적으로 보여주는데 의의가 있다.

(3) 프로그래머는 시각적인 예외 전파 정보를 이용하여 프로그램에서 발생 가능한 예외의 전파 및 처리 과정을 미리 살펴 볼 수 있으며 이를 통하여 예외를 보다 적절하게 처리하는 프로그램을 개발함으로써 결과적으로 프로그램 신뢰성 향상에 기여할 수 있다.

5. 결 론

본 논문에서는 예외 전파 경로들을 추정하는 정적 분석을 제시하고 이 정적 분석 정보를 이용해서 예외 전

파 경로를 시각적으로 보여주는 시각화 시스템을 구현하였다. 이 시스템은 프로그래머로 하여금 처리되지 않는 예외들을 찾아내고 처리되지 않는 예외들을 정확하게 선언할 수 있도록 도움을 준다. 또한 예외 전파 경로를 따라가면서 프로그래머는 예외 처리기를 놓을 적당한 위치를 찾을 수 있다.

이 시스템은 *throw*문, *try-catch*절, 그리고 메소드 선언들과 같은 예외 관련 구조들에 레이블을 기록함으로써 예외 전파 경로들을 추적하였다. 만약 좀 더 자세한 전파 정보가 필요하다면 메소드 호출과 *try* 블록들과 같은 다른 표현식들의 레이블을 포함하기 위해 예외 전파 분석을 확장할 수 있다. 또한 예외 전파 분석 정보를 이용하여 예외를 고려한 제어 흐름 그래프를 구성하거나 예외 관련된 프로그램 부분만을 추출하는데 사용될 수 있을 것이다.

참 고 문 헌

[1] B.-M. Chang, J. Jo, K. Yi, and K. Choe, "Interprocedural Exception Analysis for Java", Proceedings of ACM Symposium on Applied Computing, pp 620-625, Mar. 2001.

[2] M. P. Robillard and G. C. Murphy, Analyzing exception flow in Java programs, in Proc. of '99 European Software Engineering Conference and ACM SIGSOFT Symposium on Foundations of Software Engineering, pp. 322-337, Springer-Verlag.

[3] K. Yi and B.-M. Chang Exception analysis for Java, ECOOP Workshop on Formal Techniques for Java Programs, June 1999, Lisbon, Portugal.

[4] S. Sinha and M. Harrold, Analysis and testing of programs with exception-handling constructs, IEEE Transactions on Software Engineering 26(9) (2000)

[5] Barat, http://www.sharemation.com/~bokowski/ba rat/index.html.

[6] S. Koleth , M. Hansen, R. Zsolt, Jipe, http://jipe.sourceforge.net.

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

[8] S. Drossopoulou, and S. Eisenbach, "Java is type safe-probably", Proceedings of 97 European Conference on Object-Oriented Programming, 1997.

[9] S. Drossopoulou, and T. Valkevych, "Java type soundness revisited". Techical Report, Imperial College, November 1999. Also available from: http://www-doc.ic.ac.uk/~scd.

허 순 희

1999년 숙명여자대학교 전산학과(학사) 2002년 숙명여자대학교 컴퓨터과학과(석사). 관심분야는 객체지향프로그래밍, Program analysis, Software Engineering, 모바일 프로그래밍

창 병 모

1998년 서울대학교 컴퓨터공학과 졸업(공학사). 1990년 한국과학기술원 전산학과(공학석사). 1994년 한국과학기술원 전산학과(공학박사). 1994년~1995년 한국전자통신연구소 박사후 연수 연구원 1995년~현재 숙명여자대학교 컴퓨터과학과 부교수. 관심분야는 컴파일러 구성론(정적분석, 코드최적화), 논리 프로그래밍, 모바일 프로그래밍