예외사항을 처리하는 의미 구조 비교

한정란
협성대학교 경영정보학과
e-mail:jlhan@uhs.ac.kr

Comparison of Semantics for Handling Exceptions

Junglan Han
Dept of Management and Information System, Hyupsung University

요 약

프로그램을 실행하는 동안 다양한 오류가 발생할 수 있고 이러한 예외사항은 체계적으로 검사하고 처리해야한다. 예외사항의 의미구조를 잘 표현하면 프로그래밍언어의 신뢰성을 증진할 수 있고, 설계와 표준화, 최적화 및 번역 과정에 중요한 역할을 수행한다.

본 연구에서는 기존에 제시된 작용식을 사용하여 예외사항을 처리하는 방법을 자바 프로그램의 예제를 통하여 제시한다. 다른 의미구조들과 제시된 작용식을 판독성(Readability), 지능성(Intelligibility), 모듈성(Modularity), 확장성(Extensibility), 융통성(Flexibility)의 다섯 영역에서 비교하여 본 작용식의 우수성을 확인하고자 한다.

1. 서론

프로그램을 실행하는 동안 예기치 못한 다양한 오류가 발생할 수 있고 이러한 예외사항은 체계적으로 검사하고 처리해야한다. 예외사항의 의미구조를 잘 표현하면 보다 신뢰성 있게 프로그래밍 언어를 설계하거나 언어의 표준화와 최적화 및 번역 과정에 중요한 역할을 수행할 수 있다.

본 연구에서는 기존에 제시된 작용식[1]을 사용하여 예외사항을 처리하는 방법들을 자바 프로그램의 예제를 통하여 제시한다. 제시된 작용식을 판독성(Readability), 지능성(Intelligibility), 모듈성(Modularity), 확장성(Extensibility), 융통성(Flexibility)의 다섯 영역에서 다른 의미구조들과 비교하여 본 작용식의 우수성을 확인하고자 한다.

2. 예외사항 처리 작용식

작용식(action equation)은 언어의 동적 의미 구조를 표현하기 위한 식으로 연산 의미론(operational semantics)의 범주에 속한다[1]. 작용식 중에서 각 명령문을 실행할 동적 의미 구조를 표현하고 순차적으로 실행되는 절차적인 작용(action)인 Execute equation을 사용하여 동적 의미 구조를 표현한다.

자바언어의 처리와 관련된 가장 기본적인 작용식과 예외사항을 발생하는 작용식을 명세하면 그림 1과 같다[1]. 예외사항을 처리하는 나머지 작용식에 대한 명세는 기존연구[1]에 자세히 명세되어 있다.

◀Execute [stmts] → **event** ⟦ complete | diverge ⟧

• Execute [<s_1>,<s_2>,... <s_n> where n ≥1] →
  s_1.out ← Execute [<s_1>]
  s_2.out ← Execute [<s_2>]
  ...
  s_n.out ← Execute [<s_2>]

• Execute[**throw new** <excep_class_id>(<par_1>,<par_2>,...,<par_n>) where n≥1]→
  ano_id.env ← excep_class_id.name
  ano_id.type ← lookup2(ano_id.env, ano_id.name)
  make_table(ano_id.name, ano_id.type, ano_id.env)
  member_identifier.name ←
    search_member(excep_class_id.name)
  for each member_identifier do
    make_table(member_id.name, member_id.type ano_id.name)
  do
  ano_id.addr ← current_point
  for i=1 to n do
    $param_i$.env ← excep_class_id.env
    $param_i$.val ← Eval_out[<$param_i$>]
  od
  return_save(excep_class_id.addr)
  if (lookup_excep(excep_class_id.name)) then

```
    excep_class_id.val ← true
    excep_class_id.addr ← lookup_addr(excep_class_id.name)
  endif
  if (excep_class_id.val) then
     control_transfer(excep_class_id.addr)
  else control_transfer(NullPointerException.addr)
  endif
```

(그림 1) 예외 발생 작용식

예외사항을 처리하는 작용식을 정확하게 명세하기 위해 세 가지 모듈 **make_exception_table, lookup_excep, lookup_addr**을 정의하여 의미구조를 표현하고 예외사항을 처리하는 작용식을 보다 구체적이고 실제적으로 명세한다[1]. **make_exception_table** 모듈은 자바의 **throws** 구절을 통해 프로그램을 실행하는 도중에 발생할 수 있는 예외사항을 명시했을 경우 그 예외사항들의 목록인 **exception_table**을 작성하는 모듈로 발생 가능한 예외사항을 처리할 주소를 **catch** 구문에서 찾아 기록하는 모듈이다. **exception_table**은 **예외사항 이름**, **라인정보**, **예외사항 발생** 및 **예외사항 처리**의 네 필드로 구성된다[1]. **"예외사항 이름"**은 발생 가능한 예외사항들을 기술하는 필드이고, **"라인정보"**는 예외사항을 처리할 catch 구문의 라인 번호를 나타내고, **"예외사항 발생"**은 예외사항 발생 여부를 나타내는 필드이고, **"예외사항 처리"**는 발생한 예외사항의 처리여부를 나타내는 필드이다. **lookup_excep** 모듈은 시스템에서 내장된 예외사항과 **exception_table** 목록에서 예외사항을 찾아 그 예외사항이 발생했는지를 반환하는 모듈이다. **lookup_addr** 모듈은 **exception_table**에서 발생한 예외사항을 찾아 예외사항이 처리될 프로그램의 라인 번호를 가져와서 예외사항을 처리하기 위해 필요한 위치정보(주소)를 반환하는 모듈이다. 프로그램의 라인정보를 통해 그 라인의 상대주소를 계산할 수 있고 프로그램의 시작 주소를 알면 실행시 절대주소를 계산할 수 있다.

3. 예외사항 처리 방법

그림 2의 자바 구문에 대해 **exception_table** 목록을 작성하면서 예외사항을 처리하는 방법을 제시한다. 학과명을 입력할 때 숫자나 기타문자를 첫문자로 입력할 경우 예외사항이 발생하여 예외사항을 처리하는 구문으로 이동하여 처리하게 된다.

```
import java.io.*;
public class InputException {
public static void main(String[] args)
{ byte name[] = new byte[20];
   :
  System.out.println("학과를 입력하세요 ");
  try {
   System.in.read(name);
   String str=new String(name);
   ch=str.charAt(0);
15:  if (ch < 'A' || ch > 'z') throw new IOException();
16:  System.out.print("학과명 : ");
17:  System.out.println(str);          }
18:  catch (IOException e) {
19:   if (ch >='0' && ch <='9')
20:    msg="시작문자 "+String.valueOf(ch) +" 는 숫자입니다.";
21:   else
22:    msg="시작문자 "+String.valueOf(ch) +" 는 기타문자입니다.";
23:   System.out.println(msg);
24:   System.out.println("잘못 입력되어 "+e.toString()+" 발생");
25:   } catch(ExceptionType2 identifier) {
26:    //...
27:     :
30:   } catch(ExceptionTypen identifier) {
        //...
   } } }
```

(그림 2) 자바 프로그램

<표 1 > **exception_table** 목록

예외 이름	라인 번호	예외발생	예외 처리
IOException	18	true	true
$ExceptionType_2$	25	false	false
...	...	false	false
$ExceptionType_n$	30	false	false

표 1에 예외사항을 처리하기 위해 **exception_table** 을 작성한 것을 볼 수 있다. 첫 문자로 숫자나 기타문자가 입력했을 경우 예외사항이 발생한 것을 알 수 있다. 그림 1의 if (lookup_excep(excep_class_id.name)) then 구문에서 lookup_excep(excep_class_id.name)에 의해 예외사항이 발생한 것을 확인할 수 있고 excep_class_id.val ← **true** 로 excep_class_id.val 이 참이 된다. 예외사항이 처리되기 위해 catch 구문으로 이동해야 하고 catch 구문의 주소를 알아야 한다. lookup_addr(excep_class_id.name) 모듈에서 주소를 찾게 되

는데 표 1에 있는 목록에서 라인번호 필드 안에 있는 18이란 값을 가져오고 lookup_addr(excep_class_id.name)를 사용하여 프로그램의 시작주소로 절대 주소를 계산할 수 있다. 계산된 값이 아래의 식에 의해 excep_class_id.addr 에 들어간다.

excep_class_id.addr ← lookup_addr(excep_class_id.name)

control_transfer(excep_class_id.addr) 모듈에 의해 예외를 처리하는 주소로 이동하여 예외처리가 실행되는 것을 알 수 있다.

4. 의미구조 비교 분석

의미구조를 비교하기 위해 기존 연구[4,5,6]들을 중심으로 판독성(Readability), 지능성(Intelligibility), 모듈성(Modularity), 확장성(Extensibility), 융통성(Flexibility) 영역에서 세 표기법인 DS(Denotational Semantics), ASM(Abstract State Machine), AS(Action Semantics)와 제시된 작용식(AE)을 비교하였다. 기존의 세 가지 표기법에 대한 판정은 의미구조에 대해 비교하고 연구한 기존 논문[4,6]을 근거로 표시하였다.

<표 2> 의미 구조 비교

표현법	판독성	지능성	모듈성	확장성	융통성
DS	낮음	낮음	낮음	낮음	보통
ASM	보통	보통	보통	높음	높음
AS	보통	높음	높음	높음	낮음
AE	높음	높음	높음	높음	높음

본 연구에서 제시한 작용식은 다섯 가지 영역에서 모두 좋은 평가를 받고 있다. 제시된 작용 식은 연산 의미론(operational semantics)을 근거로 변수 값이나 자료형, 환경 등의 속성을 사용하여 간단하고 쉽게 동적 의미를 표시하고 있다. 형식 의미를 정확하게 정의하는 목적이 최적화나 번역기를 구현하는 과정에 사용하는 것인데 본 작용식을 사용하여 구현된 번역기[2]를 통해 알 수 있듯이 제시된 작용식은 번역기를 쉽게 만들 수 있도록 보다 실제적이고, 구체적이고, 정확한 의미 구조로 명세된 사실을 확인할 수 있다.

판독성, 지능성, 모듈성, 확장성, 융통성의 다섯 영역에서 작용식을 비교하면 먼저, 동적 의미 구조를 표현할 때 **if**나 **for**같은 일반적으로 널리 알려진 제어 구문을 사용하여 동적 의미 구조를 표현하여 판독성을 높일 수 있다. 자바에 대한 의미 구조를 보다 구체적이고 정확하게 명세하기 위해 make_exception_table, lookup_excep, lookup_addr 등의 모듈을 사용하여 지능성과 모듈성을 높이고 있다. 객체 처리나 예외사항 처리 등을 포함하여 언어를 확장했을 때 작용식의 경우 추가의 모듈을 정의하여 동적 의미 구조를 정확하게 명세할 수 있다. 작용식은 간단한 객체 지향언어에 대해 의미 구조를 명세할 수 있고 자바와 같은 복잡한 객체 지향 언어의 의미 구조를 표현할 수 있어 확장가능성이 높은 의미 구조 명세법이다. 다른 객체 지향 언어 구문을 명세할 경우 작용식의 문법 구문만 변경하여 그 언어의 동적 의미 구조를 쉽고 적절하게 명세할 수 있어 융통성 있는 의미 구조 명세법이라는 사실을 알 수 있다.

5. 결론

본 논문에서는 정적이고 동적인 의미 구조를 명세하는 작용식을 제시하였다. 작용식은 명령형 언어뿐만 아니라 객체 지향 언어인 자바를 위한 정적이고 동적인 의미 구조를 정확하게 명세하고 있다. 본 논문에서 제시된 작용식은 연산 의미론에 근거하여 의미구조를 정의하여 기존의 표현법과 비교할 때 아주 간단하게 의미 구조를 명세할 수 있고 번역기를 쉽게 구현할 수 있도록 보다 실제적이고 구체적이고 정확한 동적 의미구조를 명세하고 있다.

자바 언어에 대해 세 가지 의미구조 표기법인 DS, ASM, AS와 제시된 작용식(AE)을 판독성, 지능성, 모듈성, 확장성, 융통성 영역에서 비교했다. 작용식의 경우, 일반적인 제어 구문인 if 나 for 문을 사용하여 판독성을 높이고 있고, make_exception_table, lookup_excep, lookup_addr 등의 모듈을 사용하여 지능성과 모듈성을 향상시키고 있다. 추가의 모듈을 정의하여 확장된 동적 의미 구조를 명세할 수 있고, 작용식의 문법 구문만 변경함으로써 다른 언어의 동적 의미 구조를 명세할 수 있어 융통성이 높은 의미 구조이다. 따라서, 제시된 작용식이 기존의 의미 구조 표현법보다 우수함을 확인할 수 있다.

참고문헌

[1] 한정란, “예외처리를 위한 형식의미론”, 정보처리학회 논문지, 제17-A권 4호, pp. 173~180, 2010

[2] 한정란, 최성 “동적 의미 분석에 의한 점진 해석기 구축”, 인터넷정보학회 논문지, 제5권 6호 pp. 111~120, 2004

[3] 한정란, “객체 지향 언어를 위한 의미 명세”, 인터넷정보학회 논문지, 제8권 5호, pp. 35~43, 2007

[4] Yingzhou Zhang and Baowen Xu, "A Survey of Semantic Description Frameworks for Programming Languages", ACM SIGPLAN Notices Vol. 39(3), Mar 2004.

[5] J. Alves-Foss, editor. Formal Syntax and Semantics of Java, Vol 1523 of Lecture Notes in Computer Science. Springer-Verlag

[6] David A. Watt and Deryck F. Brown, "Formalising the Dynamic Semantics of Java", 2006.