An Anlysis of Uncaught Exceptions of Java

Byeong-Mo Chang Kwangkeun Yi
chang@cs.sookmyung.ac.kr kwang@cs.kaist.ac.kr

1 Introduction

Motivation

- In Java, uncaught exceptions of a method must be specified. the current implementation of Java compiler analyzes uncaught exceptions based types and specifications. However, specifications may be missing or too broad like specifying just as `Exception`.
- In the current Java compiler, Exceptions cannot propagate back into call sequence, unless they are specified. Programmers, however, often miss catching or specifying exceptions.
- Uncaught exceptions need to be analyzed more elaborately. Therefore, we will devise more elaborate uncaught exception analysis. The analysis will be presented using set-based constraints.

Main Problems

- Dynamic binding of method calls
  Variables of a class may refer to objects of its subclasses, and
  Every method is by default virtual.
  Method binding is done based on the class of the target object.
- So the first problem to be solved is
  - compile-time approximation of classes of objects, which each variable or expression refers to
  - If we consider uncaught exceptions of a method call $x.f$, then we first determine or approximate a set S of classes of objects which the variable x refers to,

Our approach

- Uncaught exceptions of a method
  - a collection of uncaught exceptions of all statements in it

Program	::=	(*ClassBody*)*
ClassBody	::=	`ClassId` ext `Classname` (*MethBody*)*
MethBody	::=	`MethId` (*FormalParameter**) [`Throws`] {*Stmts* }
Throws	::=	`throws` *ClassTypeList*
Stmts	::=	ϵ | *Stmt; Stmts*
Stmt	::=	`if` *Expr Stmt* `else` *Stmt*
	|	{ Stmts }
	|	`return` [*Expr*]
	|	*Var* := *Expr*
	|	`throw` *Expr*
	|	`try` { *Stmts* } (`catch` (*FormalParameter*) { *Stmts* })$^+$
	|	*Expr*
Expr	::=	*Value*
	|	*Var*
	|	*Expr*.`MethName`(*Expr**)
	|	new *ClassName*
	|	new *SimpleType*([*Expr*])$^+$([])*
Var	::=	*Name*
	|	*Var*.`VarName`
	|	*Var*[*Expr*]
	|	this
Value	::=	*PrimValue* | null
PrimValue	::=	*intValue* | *charValue* | *byteValue* | ...
VarType	::=	*SimpleVarType* | *ArrayType*
SimpleType	::=	*PrimType* | `ClassName` | `InterfaceName`
ArrayType	::=	*SimpleType*[] | *ArrayType*[]
PrimType	::=	bool | char | int | ...

Figure 1: Java$_s$ program

- in particular, for every method call $x.f$, we first approximate a set S of classes of objects which the variable x refers to, and for every class $c \in S$ we have to collect uncaught exceptions from $c.f$.
- we consider caught exceptions in try - catch statement
- we also consider exception passed as parameters

- presentation by set constraints

2 Programs

In this paper, we extend Java$_s$ [?] so ast to includes exception-related statements.

3 Uncaught Exception analysis

Concrete Constraints

Set variable

- V_e : a set variable for an expression e, which holds a set of values(incuding object id) that an expression e represents

  o_{id} is a newly created object id by *newc*.

- P_S : a set variable for a statement S, which holds a set of uncaught exceptions from a statement S

Abstract Constraints

- Exception is a first-class object in Java, which means exceptions can be assigned, passed or returned like other objects.

- Set variable

  - X_e : a set of classes of objects, that an expression e represents

  - P_S : a set of uncaught exceptions from a statement S

- For every expression, we need to analyse classes of objects, which each expression represents.

$$X_e \supseteq \begin{cases} \{c\} & \text{if } e \text{ contains } \texttt{new } c \\ \{X_{c.f} | c \in X_t\} & \text{if } e \text{ contains an instance field access } t.f \\ \{X_{c.f}\} & \text{if } e \text{ contains an class field access } c.f. \end{cases}$$

- for every statement S, we need to analyse uncaught exceptions

$$P_S \supseteq \begin{cases} X_e & \text{if } S \text{ is } \texttt{throw } e \\ P_e & \text{if } S \text{ contains an expression } e \\ \cup_{c \in X_e} P_{c.m\_name} & \text{if } S \text{ contains a method call } e.\texttt{m\_name} \\ P_{S_1} \cup P_{S_2} - \{E\} & \text{if } S \text{ is } \texttt{try } S_1 \texttt{ catch } (E\ x)\ S_2 \end{cases}$$

- For every method $c.m\_name$, construct the set variable

  $P_{c.m\_name} = \cup_{i=1}^{n} P_{S_i}$ if `m_name(...)` { $S_1, ..., S_n$ } is defined in a class `c`.

New
$$\overline{\texttt{new}\,c : \{V_e \supseteq \{(c, o_{id})\}\}}$$

Ass
$$\frac{\triangleright e : \mathcal{C}}{\triangleright x = e : \{V_x \supseteq V_e, P_S \supseteq P_e\} \cup \mathcal{C}}$$

Ret
$$\frac{\triangleright e : \mathcal{C}}{\triangleright \texttt{return}\, e : \{V_{c.m\_name} \supseteq V_e, P_S \supseteq P_e\} \cup \mathcal{C}} \text{ if } \texttt{return}\ e \text{ appears in } \texttt{c.m\_name}.$$

Seq
$$\frac{\triangleright S_1 : \mathcal{C}_1\ S_2 : \mathcal{C}_2}{\triangleright S_1; S_2 : \{P_{S_1;S_2} \supseteq P_1 \cup P_2\} \cup \mathcal{C}_1 \cup \mathcal{C}_2}$$

Cond
$$\frac{\triangleright e : \mathcal{C}_e\ S_1 : \mathcal{C}_1\ S_2 : \mathcal{C}_2}{\triangleright \texttt{if}\, e\, S_1 \texttt{else}\, S_2 : \{P_S \supseteq P_e \cup P_1 \cup P_2\} \cup \mathcal{C}_e \cup \mathcal{C}_1 \cup \mathcal{C}_2}$$

InstFieldAcc
$$\frac{\triangleright t : \mathcal{C}_t}{\triangleright t.f : \{V_e \supseteq V_{c.f} | (c, o_{id}) \in V_t\} \cup \mathcal{C}_t}$$

ClassFieldAcc
$$\triangleright c.f : \{V_e \supseteq V_{c.f}\}$$

throw
$$\frac{\triangleright e : \mathcal{C}}{\triangleright \texttt{throw}\, e : \{P_S \supseteq V_e \cup P_e\} \cup \mathcal{C}}$$

try
$$\frac{\triangleright S_1 : \mathcal{C}_1\ S_2 : \mathcal{C}_2}{\triangleright \texttt{try}\, S_1\ \texttt{catch}(E\ x) S2 : \{P_e \supseteq P_1 \cup P_2 - \{E\},\ V_x \supseteq P_1\} \cup \mathcal{C}_1 \cup \mathcal{C}_2}$$

MethCall
$$\frac{\triangleright e_i : \mathcal{C}_i\ i = 1, .., n}{\triangleright \begin{array}{ll} e_1.m\_name(e_2, ..., e_n) : & \{V_{x_i} \supseteq \mathcal{C}_i | i = 2, ..., n\} \cup \{V_e \supseteq V_{c.m\_name} | c \in V_{e_1}\} \cup \\ & \{P_e \supseteq P_{c.m\_name} | c \in V_{e_1}\} \cup \mathcal{C}_i \cup ... \cup \mathcal{C}_n \end{array}}$$

if $\texttt{m\_name}(x_1 : T_1, ..., x_n : T_n)\{\text{stm}\}$ is defined in a class `c`

MethBody
$$\frac{\triangleright stm : \mathcal{C}}{\triangleright mBody : \{P_{c.m\_name} \supseteq P_{stm}\} \cup \mathcal{C}}$$

if $mBody = \texttt{m\_name(...)}\{\text{stm}\}$ is defined in a class c

ClassBody
$$\frac{\triangleright mBody_i : \mathcal{C}_i,\ i = 1, ..., m}{\triangleright cBody : \mathcal{C}_1 \cup ... \cup \mathcal{C}_m}$$

if a class `C` is defined as `C ext C'` $\{mBody_1, ..., m_B ody_m\}$

Program
$$\frac{\triangleright cBody_i : \mathcal{C}_i i = 1, ..., n}{\triangleright p : \mathcal{C}_1 \cup ... \cup \mathcal{C}_n} \text{ if a program } p = cBody_1, ..., cBody_n$$

Figure 2: Constructing Concrete Semantics

New $$\frac{}{\triangleright \texttt{new} c : \{X_e \supseteq \{c\}\}}$$

Ass $$\frac{\triangleright e : \mathcal{C}}{\triangleright x = e : \{X_x \supseteq X_e, P_S \supseteq P_e\} \cup \mathcal{C}}$$

Ret $$\frac{\triangleright e : \mathcal{C}}{\triangleright \texttt{return} e : \{X_{c.m\_name} \supseteq X_e,\ P_S \supseteq P_e\} \cup \mathcal{C}}$$ if `return` e appears in `c.m_name`.

Seq $$\frac{\triangleright S_1 : \mathcal{C}_1 \quad S_2 : \mathcal{C}_2}{\triangleright S_1; S_2 : \{P_{S_1;S_2} \supseteq P_1 \cup P_2\} \cup \mathcal{C}_1 \cup \mathcal{C}_2}$$

Cond $$\frac{\triangleright e : \mathcal{C}_e \quad S_1 : \mathcal{C}_1 \quad S_2 : \mathcal{C}_2}{\triangleright \texttt{if}\ e\ S_1\ \texttt{else}\ S_2 : \{P_S \supseteq P_e \cup P_1 \cup P_2\} \cup \mathcal{C}_e \cup \mathcal{C}_1 \cup \mathcal{C}_2}$$

InstFieldAcc $$\frac{\triangleright t : \mathcal{C}_t}{\triangleright t.f : \{X_e \supseteq X_{c.f} | c \in X_t\} \cup \mathcal{C}_t}$$

ClassFieldAcc $$\frac{}{\triangleright c.f : \{X_e \supseteq X_{c.f}\}}$$

throw $$\frac{\triangleright e : \mathcal{C}}{\triangleright \texttt{throw} e : \{P_S \supseteq X_e \cup P_e\} \cup \mathcal{C}}$$

try $$\frac{\triangleright S_1 : \mathcal{C}_1 \quad S_2 : \mathcal{C}_2}{\triangleright \texttt{try}\ S_1\ \texttt{catch}(E\ x)\ S_2 : \{P_S \supseteq P_1 \cup P_2 - \{E\},\ X_x \supseteq P_1\} \cup \mathcal{C}_1 \cup \mathcal{C}_2}$$

MethCall $$\frac{\triangleright e_i : \mathcal{C}_i,\ i = 1, .., n}{\triangleright\ \begin{array}{ll} e_1.m\_name(e_2, ..., e_n) : & \{X_{x_i} \supseteq \mathcal{C}_i | i = 2, ..., n\} \cup \{X_e \supseteq X_{c.m\_name} | c \in X_{e_1}\} \cup \\ & \{P_e \supseteq P_{c.m\_name} | c \in X_{e_1}\} \cup \mathcal{C}_i \cup ... \cup \mathcal{C}_n \end{array}}$$

if `m_name`$(T x_1 : T_1, ..., x_n : T_n)\{...\ \}$ is defined in a class c

MethBody $$\frac{\triangleright stm : \mathcal{C}}{\triangleright mBody : \{P_{c.m\_name} \supseteq P_{stm}\} \cup \mathcal{C}}$$

if mBody = `m_name`(...){stm} is defined in a class `c`

ClassBody $$\frac{\triangleright mBody_i : \mathcal{C}_i, i = 1, ..., m}{\triangleright cBody : \mathcal{C}_1 \cup ... \cup \mathcal{C}_m}$$

if a class `c` is defined as `c ext c'` $\{mBody_1, ..., m_Body_m\}$

Program $$\frac{\triangleright cBody_i : \mathcal{C}_i,\ i = 1, ..., n}{\triangleright p : \mathcal{C}_1 \cup ... \cup \mathcal{C}_n}$$ if a program $p = cBody_1, ..., cBody_n$.

Figure 3: Constructing Abstract Constraints

4 Related works

Just thinking !! **Abstract Constraints using optional specifications** Every contraint is the same, except

$$MethCall \quad \frac{\triangleright e_i : \mathcal{C}_i \; i = 1, .., n}{\triangleright e_1.m\_name(e_2, ..., e_n) : \{X_{x_i} \supseteq \mathcal{C}_i | i = 1, ..., n\} \cup \{X_e \supseteq X_{c.m\_name},\; P_e \supseteq \{E_1, ..., E_n\}\} \cup \mathcal{C}_1 \cup ... \cup \mathcal{C}_n}$$

if $c \in X_{e_1}$, c.`m_name(...)` `throws` $E_1, ..., E_n\{...\}$, and x_i are formal parameters of `c.m_name`

$$MethCall \quad \frac{\triangleright e_i : \mathcal{C}_i \; i = 1, .., n}{\triangleright e_1.m\_name(e_2, ..., e_n) : \{X_{x_i} \supseteq \mathcal{C}_i | i = 1, ..., n\} \cup \{X_e \supseteq X_{c.m\_name} P_e \supseteq P_{c.m\_name}\} \cup \mathcal{C}_1 \cup ... \cup \mathcal{C}_n}$$

if $c \in X_{e_1}$, c.`m_name(...)` $\{...\}$, and x_i are formal parameters of `c.m_name`

The current Java compiler: abstract constraints using types and specifications

- Assume that every varible and expression are known its type.
- A variable $X_{c.f}$ is for the type of a variable $c.f$, that is, a variable f in a class c. Note that this holds a single type.
- $X_{c.f} = \{c'\}$ if $c.f$ is of class c'.
- for a method call, it includes specifications of uncaught exceptions in the method.

5 Conclsion

New
$$\frac{}{\texttt{new}c : \{c\}}$$

Ass
$$\frac{\triangleright e : \mathcal{C}}{\triangleright x = e : \{P_S \supseteq P_e\} \cup \mathcal{C}}$$

Ret
$$\frac{\triangleright e : \mathcal{C}}{\triangleright \texttt{return}\, e : \{P_S \supseteq P_e\} \cup \mathcal{C}}$$

Seq
$$\frac{\triangleright S_1 : \mathcal{C}_1\ S_2 : \mathcal{C}_2}{\triangleright S_1; S_2 : \{P_{S_1;S_2} \supseteq P_1 \cup P_2\} \cup \mathcal{C}_1 \cup \mathcal{C}_2}$$

Cond
$$\frac{\triangleright e : \mathcal{C}_e\ S_1 : \mathcal{C}_1\ S_2 : \mathcal{C}_2}{\triangleright \texttt{if}\, e\, S_1\ \texttt{else}\, S_2 : \{P_S \supseteq P_e \cup P_1 \cup P_2\} \cup \mathcal{C}_e \cup \mathcal{C}_1 \cup \mathcal{C}_2}$$

InstFieldAcc
$$\frac{\triangleright t : \mathcal{C}_t}{\triangleright t.f : \{X_e \supseteq X_{c.f} | c = X_t\} \cup \mathcal{C}_t}$$

ClassFieldAcc
$$\frac{}{\triangleright c.f : \{X_e \supseteq X_{c.f}\}}$$

throw
$$\frac{\triangleright e : \mathcal{C}}{\triangleright \texttt{throw}\, e : \{P_S \supseteq X_e \cup P_e\} \cup \mathcal{C}}$$

try
$$\frac{\triangleright S_1 : \mathcal{C}_1\ \ S_2 : \mathcal{C}_2}{\triangleright \texttt{try}\, S_1\ \texttt{catch}(E\, x)\, S2 : \{P_S \supseteq P_1 \cup P_2 - \{E\},\ X_x \supseteq \{E\}\} \cup \mathcal{C}_1 \cup \mathcal{C}_2}$$

MethCall
$$\frac{\triangleright}{\triangleright e_1.m\_name(e_2, ..., e_n) : \{X_e \supseteq X_{c.m\_name},\ P_e \supseteq \{E_1, ..., E_n\}\} \cup \mathcal{C}_1 \cup ... \cup \mathcal{C}_n}$$

if $X_{e_1} = c$ and `m_name(...)` `throws` $E_1, ..., E_n$ {stm} is defined in c

MethBody
$$\frac{\triangleright stm : \mathcal{C}}{\triangleright mBody : \{X_{c.m\_name} \supseteq \{T\},\ P_{c.m\_name} \supseteq P_{stm}\} \cup \mathcal{C}}$$

if `T m_name(...)` ... {stm} is defined in a class c

ClassBody
$$\frac{\triangleright mBody_i : \mathcal{C}_i i = 1, ..., m}{\triangleright cBody : \mathcal{C}_1 \cup ... \cup \mathcal{C}_m}$$

if a class `c` is defined as `c ext c'` $\{mBody_1, ..., m_B ody_m\}$

Program
$$\frac{\triangleright cBody_i : \mathcal{C}_i i = 1, ..., n}{\triangleright p : \mathcal{C}_1 \cup ... \cup \mathcal{C}_n} \text{ if a program } p = cBody_1, ..., cBody_n.$$

Figure 4: Constructing Abstract Constraints for the current Java compiler