Exception Analysis and Points-to Analysis

Better Together

Martin Bravenboer
LogicBlox

Yannis Smaragdakis
UMass Amherst

ISSTA 2009
International Symposium on Software Testing and Analysis
what do we do?

precise analysis of exception handling
improve precision and speed of points-to analyses
what do we do?

- precise analysis of exception handling
- improve precision and speed of points-to analyses

how do we do it?

- fully declarative specification
- modular extension
what do we do?
precise analysis of exception handling
improve precision and speed of points-to analyses

how do we do it?
fully declarative specification
modular extension

why do you care?
fast, sophisticated, simple
different
overview

what do we do?
precise analysis of exception handling
improve precision and speed of points-to analyses

how do we do it?
fully declarative specification
modular extension

why do you care?
fast, sophisticated, simple
different

why is it relevant?
major new experimental findings
state-of-the-art points-to analyses
what is exception analysis?

**computation of control-flow induced by exceptions**

```java
void foo() {
    if(...) {
        throw new FooException();
    }
}

void mid() {
    foo();
}

void bar() {
    try {
        foo();
    } catch(Exception exc) { ... }
}
```
what is exception analysis?

computation of control-flow induced by exceptions

```java
void foo() {
    if(...)  
        throw new FooException();
}

void mid() {
    foo();
}

void bar() {
    try {
        foo();
    }
    catch(Exception exc) {...}
}
```

- exception-flow induces interprocedural assignments
- exceptions are normal objects
- arbitrary expressions can be thrown
What is exception analysis?

Computation of control-flow induced by exceptions

```java
void foo() {
    if(...) 
        throw new FooException();
}

void mid() {
    foo();
}

void bar() {
    try {
        foo();
    } catch(SomeException e) {
        --throw e.getCause();
    }
    catch(Exception exc) {...}

    throw createSomeException();
}
```

- Exception-flow induces interprocedural assignments
- Exceptions are normal objects
- Arbitrary expressions can be thrown
what is exception analysis?

computation of control-flow induced by exceptions

```java
void foo() {
    if(...) 
        throw new FooException();
}

void mid() {
    foo();
}

void bar() {
    try {
        foo();
    } 
    catch(Exception exc) {...}
}
```

questions answered:

- what exceptions may `foo` throw?
- where may the `FooException` thrown in `foo` get caught?
- what exceptions may get caught by the handler in `bar`?
**application: program understanding**

- understand exception-flow in codebases
- coding assistance tool
- also for languages with declared checked exceptions
  - unchecked exceptions
  - `throws`-clause specifies superset
  - e.g. `IOException`

$\Rightarrow$ exception types do not explain where exceptions originate
why exception analysis? (2)

application: test coverage of exceptional situations [Fu et al.]

application

library
application: test coverage of exceptional situations [Fu et al.]
application: test coverage of exceptional situations [Fu et al.]
why exception analysis? (2)

application: test coverage of exceptional situations [Fu et al.]

![Diagram showing test suite, application, and library with arrows indicating flow and points representing test coverage.](image)
why exception analysis? (2)

application: test coverage of exceptional situations [Fu et al.]
why exception analysis? (2)

application: test coverage of exceptional situations [Fu et al.]
application: test coverage of exceptional situations [Fu et al.]
why exception analysis? (3)

points-to analysis (facilitate other applications)
what objects can a variable point to?

```plaintext
program

void foo() {
    a = new A1();
    b = id(a);
}

void bar() {
    a = new A2();
    b = id(a);
}

A id(A a) {
    return a;
}
```
what objects can a variable point to?

program

```java
void foo() {
    a = new A1();
b = id(a);
}

void bar() {
    a = new A2();
b = id(a);
}

A id(A a) {
    return a;
}
```

points-to

```plaintext
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>foo:a</td>
<td>new A1()</td>
</tr>
<tr>
<td>bar:a</td>
<td>new A2()</td>
</tr>
</tbody>
</table>
```

objects represented by allocation sites
what objects can a variable point to?

**program**

```java
void foo() {
    a = new A1();
    b = id(a);
}

void bar() {
    a = new A2();
    b = id(a);
}

A id(A a) {
    return a;
}
```

**points-to**

```
foo:a | new A1()
bar:a | new A2()
id:a  | new A1(), new A2()
```

objects represented by allocation sites
what objects can a variable point to?

**Program**

```java
void foo() {
    a = new A1();
    b = id(a);
}

void bar() {
    a = new A2();
    b = id(a);
}

A id(A a) {
    return a;
}
```

**Points-to**

```
foo:a | new A1()
bar:a | new A2()
id:a  | new A1(), new A2()
foo:b | new A1(), new A2()
bar:b | new A1(), new A2()
```

objects represented by allocation sites
what objects can a variable point to?

program

```java
void foo() {
    a = new A1();
    b = id(a);
}

void bar() {
    a = new A2();
    b = id(a);
}

A id(A a) {
    return a;
}
```

points-to

- `foo:a` | new A1()
- `bar:a` | new A2()
- `id:a` (foo) | new A1(), new A2()
- `id:a` (bar) | new A1(), new A2()
- `foo:b` | new A1(), new A2()
- `bar:b` | new A1(), new A2()

context-sensitive points-to

- `foo:a` | new A1()
- `bar:a` | new A2()
- `id:a` (foo) | new A1()
- `id:a` (bar) | new A2()
- `foo:b` | new A1()
- `bar:b` | new A2()
points-to analysis

• necessity: sound points-to analyses need to handle all language constructs

• exception analysis is different, and complicates points-to algorithms

workaround: imprecise exception analysis

\[
\text{throw } e; \quad \Rightarrow \quad \text{THROWN\_EXCEPTIONS} = e;
\]

\[
\text{catch(} \text{Exception } e); \quad \Rightarrow \quad e = \text{THROWN\_EXCEPTIONS} ;
\]

sneak preview: our finding

imprecise exception handling dominates the output of precise context-sensitive points-to analysis
program analysis: a domain of mutual recursion

- call graph analysis
- exception analysis
- points-to analysis
program analysis: a domain of mutual recursion

- call graph analysis
- points-to analysis
- exception analysis

x = y
program analysis: a domain of mutual recursion

points-to analysis

call graph analysis

exception analysis

x = y
program analysis: a domain of mutual recursion

- call graph analysis
- points-to analysis
- exception analysis

\[ x = f() \]
program analysis: a domain of mutual recursion

- call graph analysis
- exception analysis
- points-to analysis

x = f()
program analysis: a domain of mutual recursion

points-to analysis

call graph analysis

exception analysis

\[ x = y.f() \]
program analysis: a domain of mutual recursion

points-to analysis

call graph analysis

exception analysis

\[ x = y.f() \]
program analysis: a domain of mutual recursion

- call graph analysis
- points-to analysis
- exception analysis

throw e
program analysis: a domain of mutual recursion

- call graph analysis
- points-to analysis
- exception analysis

throw e
program analysis: a domain of mutual recursion

- call graph analysis
- exception analysis
- points-to analysis

catch(E e)
program analysis: a domain of mutual recursion

points-to analysis

call graph analysis

exception analysis

catch(E e)
Program analysis: a domain of mutual recursion

points-to analysis

- call graph analysis
- exception analysis

\texttt{g()}
Program analysis: a domain of mutual recursion

- Call graph analysis
- Exception analysis
- Points-to analysis

$g()$
approximation to avoid mutual recursion

- call graph analysis
- points-to analysis
- exception analysis

approximation
use conservative
approximation to avoid mutual recursion

- call graph analysis
- points-to analysis
- exception analysis

approximation

use conservative
approximation to avoid mutual recursion

- call graph analysis
- points-to analysis
- exception analysis

approximation → use conservative
approximation to avoid mutual recursion

- call graph analysis
- points-to analysis
- exception analysis

approximation

use conservative
appr oximation to avoid mutual recursion

- call graph analysis
- points-to analysis
- exception analysis
- approximation
- use conservative
approximation to avoid mutual recursion

- call graph analysis
- points-to analysis
- exception analysis

approximation

use conservative
joint exception analysis and points-to analysis

- major improvement in overall precision
- major performance improvement

where is the magic?

- our approach: no imperative algorithm, only declarative specification
- simple declarative specification of highly complex mutually recursive dependencies in datalog
source

```java
a = new A();
b = new B();
c = new C();
a = b;
b = a;
c = b;
```
source

| a = new A(); |
| b = new B(); |
| c = new C(); |
| a = b; |
| b = a; |
| c = b; |

AssignObjectAllocation

| a | new A() |
| b | new B() |
| c | new C() |

Assign

| b | a |
| a | b |
| b | c |
### datalog: declarative mutual recursion

#### source
```
a = new A();
b = new B();
c = new C();
a = b;
b = a;
c = b;
```

#### AssignObjectAllocation
```
a | new A()
b | new B()
c | new C()
```

#### Assign
```
b | a
a | b
b | c
```

#### VarPointsTo(?var, ?obj) <-
  AssignObjectAllocation(?var, ?obj).

#### VarPointsTo(?to, ?obj) <-
  Assign(?from, ?to),
  VarPointsTo(?from, ?obj).
### datalog: declarative mutual recursion

```java
a = new A();
b = new B();
c = new C();
a = b;
b = a;
c = b;
```

#### AssignObjectAllocation

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a</code></td>
<td><code>new A()</code></td>
<td></td>
</tr>
<tr>
<td><code>b</code></td>
<td><code>new B()</code></td>
<td></td>
</tr>
<tr>
<td><code>c</code></td>
<td><code>new C()</code></td>
<td></td>
</tr>
</tbody>
</table>

#### Assign

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>b</code></td>
<td><code>a</code></td>
<td></td>
</tr>
<tr>
<td><code>a</code></td>
<td><code>b</code></td>
<td></td>
</tr>
<tr>
<td><code>b</code></td>
<td><code>c</code></td>
<td></td>
</tr>
</tbody>
</table>

#### VarPointsTo

- `VarPointsTo(\?var, \?obj) \leftarrow AssignObjectAllocation(\?var, \?obj).`
- `VarPointsTo(\?to, \?obj) \leftarrow Assign(\?from, \?to), VarPointsTo(\?from, \?obj).`
source

```java
a = new A();
b = new B();
c = new C();
a = b;
b = a;
c = b;
```

AssignObjectAllocation

<table>
<thead>
<tr>
<th>var</th>
<th>obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>new A()</td>
</tr>
<tr>
<td>b</td>
<td>new B()</td>
</tr>
<tr>
<td>c</td>
<td>new C()</td>
</tr>
</tbody>
</table>

Assign

<table>
<thead>
<tr>
<th>from</th>
<th>to</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
</tr>
</tbody>
</table>

**VarPointsTo**

\[
\text{VarPointsTo}(\text{\texttt{?var}}, \text{\texttt{?obj}}) \leftarrow \text{VarPointsTo}(\text{\texttt{?to}}, \text{\texttt{?obj}}) \\
\text{VarPointsTo}(\text{\texttt{?to}}, \text{\texttt{?obj}}) \leftarrow \text{Assign}(\text{\texttt{?from}}, \text{\texttt{?to}}), \text{VarPointsTo}(\text{\texttt{?from}}, \text{\texttt{?obj}}).
\]
source

\[\begin{align*}
a &= \texttt{new} \ A(); \\
b &= \texttt{new} \ B(); \\
c &= \texttt{new} \ C(); \\
a &= b; \\
b &= a; \\
c &= b;
\end{align*}\]

AssignObjectAllocation

\[
\begin{array}{cc}
a & \texttt{new} \ A() \\
b & \texttt{new} \ B() \\
c & \texttt{new} \ C()
\end{array}
\]

VarPointsTo

\[
\begin{array}{ccc}
\text{VarPointsTo}(\texttt{var}, \texttt{obj}) & \leftarrow & \\
& \text{AssignObjectAllocation}(\texttt{var}, \texttt{obj}).
\end{array}
\]

\[
\begin{array}{ccc}
\text{VarPointsTo}(\texttt{to}, \texttt{obj}) & \leftarrow & \\
& \text{Assign}(\texttt{from}, \texttt{to}), \\
& \text{VarPointsTo}(\texttt{from}, \texttt{obj}).
\end{array}
\]
source
a = new A();
b = new B();
c = new C();
a = b;
b = a;
c = b;

AssignObjectAllocation
a | new A()
b | new B()
c | new C()

Assign
b | a
a | b
b | c

VarPointsTo(?var, ?obj) <-
AssignObjectAllocation(?var, ?obj).

VarPointsTo(?to, ?obj) <-
Assign(?from, ?to),
VarPointsTo(?from, ?obj).
**source**

```java
a = new A();
b = new B();
c = new C();
a = b;
b = a;
c = b;
```

**AssignObjectAllocation**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>new A()</td>
</tr>
<tr>
<td>b</td>
<td>new B()</td>
</tr>
<tr>
<td>c</td>
<td>new C()</td>
</tr>
</tbody>
</table>

**Assign**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
</tr>
</tbody>
</table>

**VarPointsTo**

```
VarPointsTo(?var, ?obj) <-
AssignObjectAllocation(?var, ?obj).

VarPointsTo(?to, ?obj) <-
Assign(?from, ?to),
VarPointsTo(?from, ?obj).
```
### Source

```java
a = new A();
b = new B();
c = new C();
a = b;
b = a;
c = b;
```

### AssignObjectAllocation

<table>
<thead>
<tr>
<th>var</th>
<th>object</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>new A()</td>
</tr>
<tr>
<td>b</td>
<td>new B()</td>
</tr>
<tr>
<td>c</td>
<td>new C()</td>
</tr>
</tbody>
</table>

### Assign

<table>
<thead>
<tr>
<th>from</th>
<th>to</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
</tr>
</tbody>
</table>

### VarPointsTo

```
VarPointsTo(?var, ?obj) <-
    AssignObjectAllocation(?var, ?obj).
VarPointsTo(?to, ?obj) <-
    Assign(?from, ?to),
    VarPointsTo(?from, ?obj).
```
source

a = new A();
b = new B();
c = new C();
a = b;
b = a;
c = b;

AssignObjectAllocation

<table>
<thead>
<tr>
<th>a</th>
<th>new A()</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>new B()</td>
</tr>
<tr>
<td>c</td>
<td>new C()</td>
</tr>
</tbody>
</table>

VarPointsTo

VarPointsTo(?var, ?obj) <-
    AssignObjectAllocation(?var, ?obj).

VarPointsTo(?to, ?obj) <-
    Assign(?from, ?to),
    VarPointsTo(?from, ?obj).
**Source**

\[
\begin{align*}
a &= \text{new } A(); \\
b &= \text{new } B(); \\
c &= \text{new } C(); \\
a &= b; \\
b &= a; \\
c &= b;
\end{align*}
\]

**AssignObjectAllocation**

\[
\begin{align*}
a &\quad \text{new } A() \\
b &\quad \text{new } B() \\
c &\quad \text{new } C()
\end{align*}
\]

**VarPointsTo**

\[
\begin{align*}
a &\quad \text{new } A() \\
b &\quad \text{new } B() \\
c &\quad \text{new } C()
\end{align*}
\]

**Assign**

\[
\begin{align*}
b &\quad a \\
a &\quad b \\
b &\quad c
\end{align*}
\]

\[
\text{VarPointsTo}(\text{?var, ?obj}) \leftarrow \\
\text{AssignObjectAllocation}(\text{?var, ?obj}).
\]

\[
\text{VarPointsTo}(\text{?to, ?obj}) \leftarrow \\
\text{Assign}(\text{?from, ?to}), \\
\text{VarPointsTo}(\text{?from, ?obj}).
\]
source
a = new A();
b = new B();
c = new C();
a = b;
b = a;
c = b;

AssignObjectAllocation
<table>
<thead>
<tr>
<th>a</th>
<th>new A()</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>new A()</td>
</tr>
<tr>
<td>c</td>
<td>new C()</td>
</tr>
</tbody>
</table>

VarPointsTo
<table>
<thead>
<tr>
<th>a</th>
<th>new A()</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>new B()</td>
</tr>
<tr>
<td>c</td>
<td>new C()</td>
</tr>
</tbody>
</table>

Assign

VarPointsTo(?var, ?obj) <-
AssignObjectAllocation(?var, ?obj).

VarPointsTo(?to, ?obj) <-
Assign(?from, ?to),
VarPointsTo(?from, ?obj).
datalog: declarative mutual recursion

<table>
<thead>
<tr>
<th>source</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = new A();</td>
</tr>
<tr>
<td>b = new B();</td>
</tr>
<tr>
<td>c = new C();</td>
</tr>
<tr>
<td>a = b;</td>
</tr>
<tr>
<td>b = a;</td>
</tr>
<tr>
<td>c = b;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AssignObjectAllocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>c</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assign</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
</tr>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
</tbody>
</table>

VarPointsTo(?var, ?obj) <-
   AssignObjectAllocation(?var, ?obj).

VarPointsTo(?to, ?obj) <-
   Assign(?from, ?to),
   VarPointsTo(?from, ?obj).
source

a = new A();
b = new B();
c = new C();
a = b;
b = a;
c = b;

AssignObjectAllocation

<table>
<thead>
<tr>
<th>var</th>
<th>obj</th>
</tr>
</thead>
</table>
a   | new A() |
b   | new B() |
c   | new C() |

Assign

<table>
<thead>
<tr>
<th>to</th>
<th>from</th>
</tr>
</thead>
</table>
b   | a    |
a   | b    |
b   | c    |

VarPointsTo

@VarPointsTo(?var, ?obj) <-
    AssignObjectAllocation(?var, ?obj).

@VarPointsTo(?to, ?obj) <-
    Assign(?from, ?to),
    VarPointsTo(?from, ?obj).
limited logic programming

- sql with recursion
  prolog without complex terms (constructors)
- captures PTIME complexity class

strictly declarative

- as opposed to prolog
  - conjunction commutative
  - rules commutative
- increases optimization opportunities
  - enables different execution strategies
  - enables more aggressive optimization

writing datalog is less programming, more specification
Strictly Declarative Specification of Sophisticated Points-to Analyses

- performance
- scalability
- declarative specification
- no BDDs

http://doop.program-analysis.org
method invocations: propagated exceptions

```
void f() {
  ...
}
```
method invocations: propagated exceptions

```java
void f() {
    --g();
}
```

method invocations: caught exceptions

```prolog
VarPointsTo(?param, ?obj) <-
    CallGraphEdge(?invocation, ?tomethod),
    ThrowPointsTo(?tomethod, ?obj),
    Type(?obj) = ?objtype,
    ExceptionHandler(?objtype, ?invocation) = ?handler,
```
method invocations: propagated exceptions

ThrowPointsTo(?caller, ?obj) <-

Method declaration ?caller may throw exception object ?obj

void f() {
   g();
}
method invocations: propagated exceptions

\[
\text{ThrowPointsTo}(\text{?caller, ?obj}) \leftarrow \\
\text{CallGraphEdge}(\text{?invitation, ?tomenthod}), \\
\text{Object:Type}[?obj] = ?objtype, \\
\text{not exists ExceptionHandler}[?objtype, ?invitation], \\
\text{Instruction:Method}[?invitation] = ?caller.
\]

void f() {
    g();
}
method invocations: propagated exceptions

\[
\text{ThrowPointsTo}(\text{?caller}, \text{?obj}) \leftarrow \\
\text{CallGraphEdge}(\text{?invocation}, \text{?tomethod}), \\
\text{ThrowPointsTo}(\text{?tomethod}, \text{?obj}),
\]

Method declaration \text{?tomethod} may throw exception object \text{?obj}

```java
void f() {
  --g();
}
```
The type of the object allocated at ?obj is ?objtype
**method invocations: propagated exceptions**

Define a rule `ThrowPointsTo(?caller, ?obj)` if:
- There is a call graph edge `CallGraphEdge(?invocation, ?tomethod)`,
- `ThrowPointsTo(?tomethod, ?obj)`,
- `Object:Type[?obj] = ?objtype`,
- No exists `ExceptionHandler[?objtype, ?invocation]`,

```java
void f() {
    --g();
}
```

Exceptions of specific type `?objtype`, thrown at instruction `?invocation`, are handled by exception handler `?handler`.
void f() {
    g();
}

Instruction ?invocation is in method ?caller

ThrowPointsTo(?caller, ?obj) <-
   CallGraphEdge(?invocation, ?tomethod),
   ThrowPointsTo(?tomethod, ?obj),
   Object:Type[?obj] = ?objtype,
   not exists ExceptionHandler[?objtype, ?invocation],
method invocations: propagated exceptions

ThrowPointsTo(?caller, ?obj) <-
   CallGraphEdge(?invocation, ?tomethod),
   ThrowPointsTo(?tomethod, ?obj),
   Object:Type[?obj] = ?objtype,
   not exists ExceptionHandler[?objtype, ?invocation],

method invocations: caught exceptions

void f() {
   try {...}
   catch(E e) {...}
}
method invocations: propagated exceptions

\[
\text{ThrowPointsTo}(\text{?caller, ?obj}) \leftarrow \\
\text{CallGraphEdge}(\text{?invocation, ?tomethod}), \\
\text{ThrowPointsTo}(\text{?tomethod, ?obj}), \\
\text{Object:Type}[\text{?obj}] = \text{?objtype}, \\
\text{not exists ExceptionHandler}[\text{?objtype, ?invocation}], \\
\text{Instruction:Method}[\text{?invocation}] = \text{?caller}.
\]

method invocations: caught exceptions

\[
\text{VarPointsTo}(\text{?param, ?obj}) \leftarrow \\
\text{CallGraphEdge}(\text{?invocation, ?tomethod}), \\
\text{ThrowPointsTo}(\text{?tomethod, ?obj}), \\
\text{Type}[\text{?obj}] = \text{?objtype}, \\
\text{ExceptionHandler}[\text{?objtype, ?invocation}] = \text{?handler}, \\
\text{ExceptionHandler:FormalParam}[\text{?handler}] = \text{?param}.
\]

\[
\text{void f() \{ } \\
\text{\hspace{1cm} g(); } \\
\text{\} }
\]

\[
\text{void f() \{ } \\
\text{\hspace{1cm} try \{ g(); \} } \\
\text{\hspace{1cm} catch(E e) \{ ... \} } \\
\text{\}}
\]
declarative on-the-fly exception analysis

method invocations: propagated exceptions

$$\text{ThrowPointsTo}(\text{?caller, } \text{?obj}) \leftarrow$$
$$\text{CallGraphEdge}(\text{?invocation, } \text{?tomethod}),$$
$$\text{ThrowPointsTo}(\text{?tomethod, } \text{?obj}),$$
$$\text{Object:Type}[\text{?obj}] = \text{?objtype},$$
$$\text{not exists ExceptionHandler}[\text{?objtype, } \text{?invocation}],$$
$$\text{Instruction:Method}[\text{?invocation}] = \text{?caller}.$$  

```
void f() {
  --g();
}
```

method invocations: caught exceptions

$$\text{VarPointsTo}(\text{?param, } \text{?obj}) \leftarrow$$

```
void f() {
  --try { g(); }
  --catch(E e) {...}
}
```
method invocations: propagated exceptions

ThrowPointsTo(?caller, ?obj) <-
  CallGraphEdge(?invocation, ?tomethod),
  ThrowPointsTo(?tomethod, ?obj),
  Object:Type[?obj] = ?objtype,
  not exists ExceptionHandler[?objtype, ?invocation],

void f() {
  g();
}

method invocations: caught exceptions

VarPointsTo(?param, ?obj) <-
  CallGraphEdge(?invocation, ?tomethod),
  ThrowPointsTo(?tomethod, ?obj),
  Type[?obj] = ?objtype,
method invocations: propagated exceptions

\[
\text{ThrowPointsTo}(\text{?caller}, \text{?obj}) \leftarrow \\
\text{CallGraphEdge}(\text{?invocation}, \text{?tomethod}), \\
\text{ThrowPointsTo}(\text{?tomethod}, \text{?obj}), \\
\text{Object:Type}[\text{?obj}] = \text{?objtype}, \\
\text{not exists ExceptionHandler}[\text{?objtype}, \text{?invocation}], \\
\text{Instruction:Method}[\text{?invocation}] = \text{?caller}.
\]

method invocations: caught exceptions

\[
\text{VarPointsTo}(\text{?param}, \text{?obj}) \leftarrow \\
\text{CallGraphEdge}(\text{?invocation}, \text{?tomethod}), \\
\text{ThrowPointsTo}(\text{?tomethod}, \text{?obj}), \\
\text{Type}[\text{?obj}] = \text{?objtype}, \\
\text{ExceptionHandler}[\text{?objtype}, \text{?invocation}] = \text{?handler},
\]

void \text{f}() 
\{
    \text{try} \{ \text{g}(); \}
    \text{catch}(\text{E e}) \{ \ldots \}
\}
method invocations: propagated exceptions

\[
\text{ThrowPointsTo}(\text{?caller}, \text{?obj}) \leftarrow \\
\text{CallGraphEdge}(\text{?invocation}, \text{?tomethod}), \\
\text{ThrowPointsTo}(\text{?tomethod}, \text{?obj}), \\
\text{Object:Type}[\text{?obj}] = \text{?objtype}, \\
\text{not exists ExceptionHandler}[\text{?objtype}, \text{?invocation}], \\
\text{Instruction:Method}[\text{?invocation}] = \text{?caller}.
\]

method invocations: caught exceptions

\[
\text{VarPointsTo}(\text{?param}, \text{?obj}) \leftarrow \\
\text{CallGraphEdge}(\text{?invocation}, \text{?tomethod}), \\
\text{ThrowPointsTo}(\text{?tomethod}, \text{?obj}), \\
\text{Type}[\text{?obj}] = \text{?objtype}, \\
\text{ExceptionHandler}[\text{?objtype}, \text{?invocation}] = \text{?handler}, \\
\text{ExceptionHandler:FormalParam}[\text{?handler}] = \text{?param}.
\]

```java
don't get code
```
what did you just see here?

• modular extension of variety of base points-to analyses
• approximation only comes from points-to abstraction – exception logic as precise as possible!
• complex mutually recursive dependencies
• specified elegantly in a few lines of logic

you might wonder ... does that work?!
experimental findings
statistics highlights for **object sensitive** analysis:

- **precision of points-to results**
  - context-insensitive: imprecise $> \text{precise} \times 1.9$
  - context-sensitive: imprecise $> \text{precise} \times 3$

- **size of call graph**
  - context-insensitive: no significant difference
  - context-sensitive: $1.9 \times$ to $6.1 \times$ more edges

- **performance**
  - imprecise $14 \times, 12 \times, 5-10 \times, 1.8 \times$ slower
statistics highlights for **object sensitive** analysis:

- precision of points-to results
  - context-insensitive: imprecise $> \text{precise} \times 1.9$
  - context-sensitive: imprecise $> \text{precise} \times 3$

- size of call graph
  - context-insensitive: no significant difference
  - context-sensitive: $1.9\times$ to $6.1\times$ more edges

- performance
  - imprecise $14\times$, $12\times$, $5-10\times$, $1.8\times$ slower

**our finding**

*Precise exception handling has a major impact on the precision and performance of context-sensitive points-to analyses. With imprecise exception handling, the size of the problem is largely determined by exceptions.*
why exception analysis? (2)

application: test coverage of exceptional situations [Fu et al.]
test coverage: possible exception-catch links [Fu et al.]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>I/Osel</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>imprecise</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ftpd</td>
<td>insens 1 obj</td>
<td>104</td>
<td>12s</td>
</tr>
<tr>
<td>muffin</td>
<td>insens 1 obj</td>
<td>490</td>
<td>22s</td>
</tr>
<tr>
<td><strong>precise</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ftpd</td>
<td>insens 1 obj</td>
<td></td>
<td></td>
</tr>
<tr>
<td>muffin</td>
<td>insens 1 obj</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**test coverage: possible exception-catch links** [Fu et al.]

<table>
<thead>
<tr>
<th></th>
<th>I/O sel</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>imprecise</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ftpd</td>
<td>insens</td>
<td>104</td>
</tr>
<tr>
<td>1 obj</td>
<td></td>
<td>91</td>
</tr>
<tr>
<td>muffin</td>
<td>insens</td>
<td>490</td>
</tr>
<tr>
<td>1 obj</td>
<td></td>
<td>420</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>precise</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ftpd</td>
<td>insens</td>
<td>47</td>
</tr>
<tr>
<td>1 obj</td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>muffin</td>
<td>insens</td>
<td>237</td>
</tr>
<tr>
<td>1 obj</td>
<td></td>
<td>49</td>
</tr>
</tbody>
</table>
## Test Coverage: Possible Exception-Catch Links

[Fu et al.]

<table>
<thead>
<tr>
<th></th>
<th>I/Osel</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Imprecise</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ftpd</td>
<td>insens 1 obj</td>
<td>104</td>
</tr>
<tr>
<td>muffin</td>
<td>insens 1 obj</td>
<td>490</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Precise</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ftpd</td>
<td>insens 1 obj</td>
<td>47</td>
</tr>
<tr>
<td>muffin</td>
<td>insens 1 obj</td>
<td>237</td>
</tr>
</tbody>
</table>

custom: \(\sim 5\text{min}\)

custom: \(> 1\text{h}\)
test coverage: possible exception-catch links [Fu et al.]

<table>
<thead>
<tr>
<th></th>
<th>I/O</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>imprecise</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ftpd</td>
<td>insens</td>
<td>104</td>
</tr>
<tr>
<td></td>
<td>1 obj</td>
<td>91</td>
</tr>
<tr>
<td>muffin</td>
<td>insens</td>
<td>490</td>
</tr>
<tr>
<td></td>
<td>1 obj</td>
<td>420</td>
</tr>
<tr>
<td>precise</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ftpd</td>
<td>insens</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>1 obj</td>
<td>15</td>
</tr>
<tr>
<td>muffin</td>
<td>insens</td>
<td>237</td>
</tr>
<tr>
<td></td>
<td>1 obj</td>
<td>49</td>
</tr>
</tbody>
</table>

custom: ~ 5min

custom: > 1h

our finding

*Our general joint points-to and exception analysis achieves precision comparable to a custom exception-flow analysis, but runs much faster.*
selectively remove features from fully precise analysis

- order of exception handlers not considered (o)

```java
catch (FileNotFoundException e) {...}
catch (IOException e) {...}
```

- no filtering of caught exceptions (f)

```java
void foo() {
    try {...}
    catch (IOException e) {...}
}
```

- context-insensitive throw points-to (cs)
  - methods throw same exceptions in all contexts
major experimental findings: approximations

<table>
<thead>
<tr>
<th>cs</th>
<th>o</th>
<th>f</th>
<th>call graph edges</th>
<th>var points-to</th>
<th>throw points-to</th>
</tr>
</thead>
<tbody>
<tr>
<td>×</td>
<td>×</td>
<td>×</td>
<td>1.0M</td>
<td>598K</td>
<td>579K</td>
</tr>
</tbody>
</table>

Every approximation of exception handling significantly increases var points-to, throw points-to, or call graph edges.
Every approximation of exception handling significantly increases var points-to, throw points-to, or call graph edges.

<table>
<thead>
<tr>
<th>cs o f</th>
<th>call graph edges</th>
<th>var points-to</th>
<th>throw points-to</th>
</tr>
</thead>
<tbody>
<tr>
<td>×</td>
<td>1.0M</td>
<td>598K</td>
<td>579K</td>
</tr>
<tr>
<td>×</td>
<td>×1.5</td>
<td>×1.0</td>
<td>×1.1</td>
</tr>
</tbody>
</table>

imprecise
## Major Experimental Findings: Approximations

<table>
<thead>
<tr>
<th>cs o f</th>
<th>Call Graph Edges</th>
<th>Var Points-to</th>
<th>Throw Points-to</th>
</tr>
</thead>
<tbody>
<tr>
<td>× × ×</td>
<td>1.0M</td>
<td>598K</td>
<td>579K</td>
</tr>
<tr>
<td>× ×</td>
<td>×1.5</td>
<td>×1.0</td>
<td>×1.1</td>
</tr>
<tr>
<td>× ×</td>
<td>×2.6</td>
<td>×1.2</td>
<td>×1.9</td>
</tr>
</tbody>
</table>

Every approximation of exception handling significantly increases var points-to, throw points-to, or call graph edges.
Every approximation of exception handling significantly increases var points-to, throw points-to, or call graph edges.
### major experimental findings: approximations

<table>
<thead>
<tr>
<th>cs</th>
<th>o</th>
<th>f</th>
<th>call graph edges</th>
<th>var points-to</th>
<th>throw points-to</th>
</tr>
</thead>
<tbody>
<tr>
<td>×</td>
<td>×</td>
<td>×</td>
<td>1.0M</td>
<td>598K</td>
<td>579K</td>
</tr>
<tr>
<td>×</td>
<td>×</td>
<td></td>
<td>×1.5</td>
<td>×1.0</td>
<td>×1.1</td>
</tr>
<tr>
<td>×</td>
<td>×</td>
<td></td>
<td>×2.6</td>
<td>×1.2</td>
<td>×1.9</td>
</tr>
<tr>
<td>×</td>
<td></td>
<td></td>
<td>×2.6</td>
<td>×1.3</td>
<td>×1.9</td>
</tr>
<tr>
<td>×</td>
<td>×</td>
<td></td>
<td>×1.1</td>
<td>×1.1</td>
<td>×1.9</td>
</tr>
</tbody>
</table>
major experimental findings: approximations

<table>
<thead>
<tr>
<th>cs o f</th>
<th>call graph edges</th>
<th>var points-to</th>
<th>throw points-to</th>
</tr>
</thead>
<tbody>
<tr>
<td>× × ×</td>
<td>1.0M</td>
<td>598K</td>
<td>579K</td>
</tr>
<tr>
<td>× ×</td>
<td>×1.5</td>
<td>×1.0</td>
<td>×1.1</td>
</tr>
<tr>
<td>× ×</td>
<td>×2.6</td>
<td>×1.2</td>
<td>×1.9</td>
</tr>
<tr>
<td>×</td>
<td>×2.6</td>
<td>×1.3</td>
<td>×1.9</td>
</tr>
<tr>
<td>× ×</td>
<td>×1.1</td>
<td>×1.1</td>
<td>×1.9</td>
</tr>
<tr>
<td>×</td>
<td>×1.6</td>
<td>×1.2</td>
<td>×2.1</td>
</tr>
<tr>
<td>×</td>
<td>×2.7</td>
<td>×1.4</td>
<td>×3.4</td>
</tr>
<tr>
<td>×</td>
<td>×2.7</td>
<td>×1.5</td>
<td>×3.4</td>
</tr>
</tbody>
</table>

Every approximation of exception handling significantly increases var points-to, throw points-to, or call graph edges.
**major experimental findings: approximations**

<table>
<thead>
<tr>
<th>cs o f</th>
<th>call graph edges</th>
<th>var points-to</th>
<th>throw points-to</th>
</tr>
</thead>
<tbody>
<tr>
<td>✗ ✗ ✗ ✗</td>
<td>1.0M</td>
<td>598K</td>
<td>579K</td>
</tr>
<tr>
<td>✗ ✗ ✗</td>
<td>✗1.5</td>
<td>✗1.0</td>
<td>✗1.1</td>
</tr>
<tr>
<td>✗ ✗</td>
<td>✗2.6</td>
<td>✗1.2</td>
<td>✗1.9</td>
</tr>
<tr>
<td>✗</td>
<td>✗2.6</td>
<td>✗1.3</td>
<td>✗1.9</td>
</tr>
<tr>
<td>✗ ✗</td>
<td>✗1.1</td>
<td>✗1.1</td>
<td>✗1.9</td>
</tr>
<tr>
<td>✗</td>
<td>✗1.6</td>
<td>✗1.2</td>
<td>✗2.1</td>
</tr>
<tr>
<td>✗</td>
<td>✗2.7</td>
<td>✗1.4</td>
<td>✗3.4</td>
</tr>
<tr>
<td>✗</td>
<td>✗2.7</td>
<td>✗1.5</td>
<td>✗3.4</td>
</tr>
<tr>
<td>imprecise</td>
<td>✗6.1</td>
<td>✗2.0</td>
<td>-</td>
</tr>
</tbody>
</table>
### major experimental findings: approximations

<table>
<thead>
<tr>
<th>cs</th>
<th>o</th>
<th>f</th>
<th>call graph edges</th>
<th>var points-to</th>
<th>throw points-to</th>
</tr>
</thead>
<tbody>
<tr>
<td>×</td>
<td>×</td>
<td>×</td>
<td>1.0M</td>
<td>598K</td>
<td>579K</td>
</tr>
<tr>
<td>×</td>
<td>×</td>
<td></td>
<td>×1.5</td>
<td>×1.0</td>
<td>×1.1</td>
</tr>
<tr>
<td>×</td>
<td>×</td>
<td></td>
<td>×2.6</td>
<td>×1.2</td>
<td>×1.9</td>
</tr>
<tr>
<td>×</td>
<td></td>
<td></td>
<td>×2.6</td>
<td>×1.3</td>
<td>×1.9</td>
</tr>
<tr>
<td>×</td>
<td>×</td>
<td></td>
<td>×1.1</td>
<td>×1.1</td>
<td>×1.9</td>
</tr>
<tr>
<td>×</td>
<td></td>
<td></td>
<td>×1.6</td>
<td>×1.2</td>
<td>×2.1</td>
</tr>
<tr>
<td>×</td>
<td></td>
<td></td>
<td>×2.7</td>
<td>×1.4</td>
<td>×3.4</td>
</tr>
<tr>
<td>×</td>
<td></td>
<td></td>
<td>×2.7</td>
<td>×1.5</td>
<td>×3.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>×6.1</td>
<td>×2.0</td>
<td>-</td>
</tr>
</tbody>
</table>

**our finding**

*Every approximation of exception handling significantly increases var points-to, throw points-to, or call graph edges.*
Precise exception handling has a major impact on the precision and performance of context-sensitive points-to analyses.

Our general joint points-to and exception analysis achieves precision comparable to a custom exception-flow analysis, but runs much faster.

Every approximation of exception handling significantly increases var points-to, throw points-to, or call graph edges.
**related work**

**type-based exception analyses** [Robillard, Jex]
- do not determine where an exception comes from
- conservative/unsound for ‘computed’ exceptions
related work

**type-based exception analyses** [Robillard, Jex]
- do not determine where an exception comes from
- conservative/unsound for ‘computed’ exceptions

**exception-flow and exception-chain analysis** [Fu et al.]
- precise analysis
- slow, automatically supported by points-to analysis
related work

**type-based exception analyses** [Robillard, Jex]
- do not determine where an exception comes from
- conservative/unsound for ‘computed’ exceptions

**exception-flow and exception-chain analysis** [Fu et al.]
- precise analysis
- slow, automatically supported by points-to analysis

**spark, paddle** [Lhotak et al.], **bdddbdddb** [Whaley et al]
- imprecise exception analysis
- generally not integrated in the analysis
**related work**

**type-based exception analyses** [Robillard, Jex]
- do not determine where an exception comes from
- conservative/unsound for ‘computed’ exceptions

**exception-flow and exception-chain analysis** [Fu et al.]
- precise analysis
- slow, automatically supported by points-to analysis

**spark, paddle** [Lhotak et al.], **bddbddd** [Whaley et al]
- imprecise exception analysis
- generally not integrated in the analysis

**doop compared to other datalog-based points-to analysis**
- full end-to-end analysis in datalog
- first precise declarative exception analysis
what have we seen?

- joint points-to and exception analysis
what have we seen?

- joint points-to and exception analysis
- precision of exception analysis has significant impact on points-to analysis
what have we seen?

- joint points-to and exception analysis
- precision of exception analysis has significant impact on points-to analysis
- exception analysis as precise, but much faster than custom exception analyses
what have we seen?

- joint points-to and exception analysis
- precision of exception analysis has significant impact on points-to analysis
- exception analysis as precise, but much faster than custom exception analyses

what more is in the paper?

- computing exception handlers
- experiments
- background on datalog and points-to analysis