Definite Expression Aliasing Analysis
for Java Bytecode

Đurica Nikolić\textsuperscript{1,2} and Fausto Spoto\textsuperscript{1}

1. - Dipartimento di Informatica, University of Verona (Italy)
2. - Microsoft Research - University of Trento Centre for Computational and Systems Biology

Bangalore, 25\textsuperscript{th} September 2012
Motivating Example: HoneycombGallery

```java
if (mCamera != null) {
    mCamera.stopPreview();
    mPreview.setCamera(null);
    mCamera.release();
    mCamera = null;
}
else:
    ...........
```
Motivating Example: HoneycombGallery

if (mCamera != null) {
    mCamera.stopPreview();
    mPreview.setCamera(null);
    mCamera.release();
    mCamera = null;
}

else:
    ...........
**Motivating Example: HoneycombGallery**

```java
if (mCamera != null) {
    mCamera.stopPreview();
    mPreview.setCamera(null);
    mCamera.release();
    mCamera = null;
}
else:
    ...........
```

- IF THE TOP OF THE STACK IS NULL, GO TO ELSE
**Motivating Example: HoneycombGallery**

```java
if (mCamera != null) {
    mCamera.stopPreview();
    mPreview.setCamera(null);
    mCamera.release();
    mCamera = null;
} else {
    ...........

    // Code continues here...
```
**Motivating Example: HoneycombGallery**

```
if (mCamera != null) {
    THEN:
    mCamera.stopPreview();
    mPreview.setCamera(null);
    mCamera.release();
    mCamera = null;
} ELSE:
    ...........
```

- **If** the top of the stack is null, **go to** else
- **Ifnull** is definitely non-null
- **ifnull** removes the top of the stack
Motivating Example: HoneycombGallery

```java
if (mCamera != null) {
    mCamera.stopPreview();
    mPreview.setCamera(null);
    mCamera.release();
    mCamera = null;
}
```

- THEN:
  - mCamera.stopPreview();
  - mPreview.setCamera(null);
  - mCamera.release();
  - mCamera = null;

- ELSE:
  - ...........

```java
aload_0
getfield mCamera:Landroid/hardware/Camera;
ifnull ELSE
THEN:
...
```

- IF THE TOP OF THE STACK IS NULL, GO TO ELSE
- AT THEN: THE TOP OF THE STACK IS DEFINITELY NON-NULL
- ifnull REMOVES THE TOP OF THE STACK BUT IT IS ALIASED TO this.mCamera
**Motivating Example: HoneycombGallery**

```java
if (mCamera != null) {
    mCamera.stopPreview();
    mPreview.setCamera(null);
    mCamera.release();
    mCamera = null;
} else {
    ...........
}
```

- `mCamera.stopPreview()` does not launch a `NullPointerException` if the top of the stack is null, go to else
- `ifnull` is definitely non-null
- Ifnull removes the top of the stack
- But it is aliased to `this.mCamera`

- `mCamera.stopPreview()` does not launch a `NullPointerException`
Haven’t we solved this problem yet?

There is a lot of pointer analyses: [Hind01] surveys more than 75 papers.
HAVEN’T WE SOLVED THIS PROBLEM YET?

There is a lot of pointer analyses: [Hind01] surveys more than 75 papers

POSSIBLE (MAY) ALIASING

- Pairs of variables that might point to the same memory location
- Over-approximation
- WALA, soot, JAAT
HAVEN’T WE SOLVED THIS PROBLEM YET?

There is a lot of pointer analyses: [Hind01] surveys more than 75 papers

POSSIBLE (MAY) ALIASING

- Pairs of variables that might point to the same memory location
- Over-approximation
- WALA, soot, JAAT

DEFINITE (MUST) ALIASING

- Pairs of variables that must point to the same memory location
- Under-approximation
- No tool for Java or Java bytecode
Haven't we solved this problem yet?
There is a lot of pointer analyses: [Hind01] surveys more than 75 papers

POSSIBLE (MAY) ALIASING
- Pairs of variables that might point to the same memory location
- Over-approximation
- WALA, soot, JAAT

DEFINITE (MUST) ALIASING
- Pairs of variables that must point to the same memory location
- Under-approximation
- No tool for Java or Java bytecode

We provide a novel approach dealing with Java bytecode programs
and providing expressions definitely aliased to variables
WHERE CAN IT BE USEFUL?

- **EXPRESSIONS DEFINITELY ALIASED TO V ARE NON-NULL AFTER** \( \text{if } (v!=\text{null}) \)
- **EXPRESSIONS** \( E.f \) **ARE NON-NULL AFTER AN ASSIGNMENT** \( w.f=exp \) **IF**
  - exp IS NON-NULL AND
  - \( E \) IS DEFINITELY ALIASED TO \( w \) AND
  - EVALUATIONS OF \( E \) MUST NOT UPDATE \( f \)
- **INFERENCE OF SYMBOLIC UPPER AND LOWER BOUNDS AFTER COMPARISON** \( x < y \):
  - **EXPRESSIONS DEFINITELY ALIASED TO** \( y \) **ARE UPPER BOUNDS FOR** \( x \)
  - **EXPRESSIONS DEFINITELY ALIASED TO** \( x \) **ARE LOWER BOUNDS FOR** \( y \)
public class Event {
    public int hr, min;
    ...
    public int delayMinBy(int offset) {
        return min + offset;
    }
    ...
}
public class Event {
    public int hr, min;
    ...
    public int delayMinBy(int offset) {
        return min + offset;
    }
    ...
}
**STATE**

**LOCAL VARIABLES:** \( L = \{ l_0, \ldots, l_{i-1} \} \)

**STACK ELEMENTS:** \( S = \{ s_0, \ldots, s_{j-1} \} \)

**VARIABLES:** \( V = L \cup S = \{ v_1, \ldots, v_{i+j} \} \)
STATE

LOCAL VARIABLES: \( L = \{ l_0, \ldots, l_{i-1} \} \)
STACK ELEMENTS: \( S = \{ s_0, \ldots, s_{j-1} \} \)

VARIABLES: \( V = L \cup S = \{ v_1, \ldots, v_{i+j} \} \)

MEMORY \( \mu \) →

EVENT

<table>
<thead>
<tr>
<th>Event</th>
<th>( \ell_1 )</th>
<th>( o_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>hr</td>
<td>min</td>
<td>10</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ListEvents</td>
<td>( \ell_2 )</td>
<td>( o_2 )</td>
</tr>
<tr>
<td>head</td>
<td>tail</td>
<td>null</td>
</tr>
<tr>
<td>Event</td>
<td>( \ell_3 )</td>
<td>( o_3 )</td>
</tr>
<tr>
<td>hr</td>
<td>min</td>
<td>20</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ListEvents</td>
<td>( \ell_4 )</td>
<td>( o_4 )</td>
</tr>
<tr>
<td>head</td>
<td>tail</td>
<td></td>
</tr>
</tbody>
</table>

EVENT

<table>
<thead>
<tr>
<th>Event</th>
<th>( \ell_1 )</th>
<th>( o_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_0</td>
<td>v_1</td>
<td>1</td>
</tr>
<tr>
<td>l_1</td>
<td>v_2</td>
<td>20</td>
</tr>
<tr>
<td>l_2</td>
<td>v_3</td>
<td>( \ell_3 )</td>
</tr>
<tr>
<td>l_3</td>
<td>v_4</td>
<td>( \ell_4 )</td>
</tr>
<tr>
<td>s_0</td>
<td>v_5</td>
<td>( \ell_2 )</td>
</tr>
</tbody>
</table>

\( \sigma = \langle \langle [@\ell_1, 20, @\ell_3, @\ell_4] \parallel @\ell_2 \rangle, \mu \rangle \)
**DEFINITION OF EXPRESSIONS**

**EXPRESSIONS**

- $\mathcal{F}$ - SET OF ALL FIELD NAMES
- $\mathcal{M}$ - SET OF ALL METHOD NAMES

We define $\mathcal{E}$, the set of expressions:

$$\mathcal{E} \ni \mathcal{E} ::= n \text{ constants} \mid v \text{ variables} \mid \mathcal{E} \oplus \mathcal{E} \text{ arithmetic expressions} \mid \mathcal{E}.f \text{ field accesses} \mid \mathcal{E}.m(\mathcal{E},...) \text{ method invocations},$$

where $n \in \mathbb{Z}$, $v \in \mathbb{V}$, $\oplus \in \{+,-,\times,\div,\%\}$, $f \in \mathcal{F}$ and $m \in \mathcal{M}$. 

---

D. Nikolić, F. Spoto (ICTAC 2012)
**Expressions**

- $\mathcal{F}$ - set of all field names
- $\mathcal{M}$ - set of all method names

We define $\mathcal{E}$, the set of expressions:

$\mathcal{E} \ni E ::=$

- $n$ — constants
- $v$ — variables
- $E \oplus E$ — arithmetic expressions
- $E.f$ — field accesses
- $E.m(E, \ldots)$ — method invocations

Where $n \in \mathbb{Z}$, $v \in V$, $\oplus \in \{+,-,\times,\div,\%\}$, $f \in \mathcal{F}$ and $m \in \mathcal{M}$. 
## Evaluation of Expressions

**Evaluation of** $l_2.\text{min}$ in $\langle \rho, \mu \rangle$: $[[l_2.\text{min}]]\langle \rho, \mu \rangle$

### Memory $\mu$

<table>
<thead>
<tr>
<th>Event</th>
<th>Event</th>
<th>$o_1$</th>
<th>$\ell_1$</th>
<th>$o_2$</th>
<th>$\ell_2$</th>
<th>$o_3$</th>
<th>$\ell_3$</th>
<th>$o_4$</th>
<th>$\ell_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>hr</td>
<td>min</td>
<td></td>
<td></td>
<td>hr</td>
<td>min</td>
<td></td>
<td></td>
<td>hr</td>
<td>min</td>
</tr>
<tr>
<td>10</td>
<td>15</td>
<td></td>
<td></td>
<td>10</td>
<td>15</td>
<td></td>
<td></td>
<td>18</td>
<td>20</td>
</tr>
<tr>
<td>ListEvents</td>
<td>head</td>
<td>tail</td>
<td>null</td>
<td>ListEvents</td>
<td>head</td>
<td>tail</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Environment $\rho$

<table>
<thead>
<tr>
<th>$l_0$</th>
<th>$v_1$</th>
<th>$l_1$</th>
<th>$l_2$</th>
<th>$v_2$</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_3$</td>
<td>$v_3$</td>
<td>$\ell_3$</td>
<td>$l_4$</td>
<td>$v_4$</td>
<td>$l_4$</td>
</tr>
<tr>
<td>$s_0$</td>
<td>$v_5$</td>
<td>$l_2$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Evaluation of Expressions

**MEMORY \( \mu \)**

<table>
<thead>
<tr>
<th>Event</th>
<th>hr</th>
<th>min</th>
<th>o₁</th>
<th>ℓ₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListEvents</td>
<td>10</td>
<td>15</td>
<td>o₂</td>
<td>ℓ₂</td>
</tr>
<tr>
<td>head</td>
<td>tail</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Event</td>
<td>hr</td>
<td>min</td>
<td>o₃</td>
<td>ℓ₃</td>
</tr>
<tr>
<td>ListEvents</td>
<td>18</td>
<td>20</td>
<td>o₄</td>
<td>ℓ₄</td>
</tr>
<tr>
<td>head</td>
<td>tail</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ENVIRONMENT \( \rho \)**

<table>
<thead>
<tr>
<th>ℓ₀</th>
<th>v₁</th>
<th>ℓ₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>l₁</td>
<td>v₂</td>
<td>20</td>
</tr>
<tr>
<td>l₂</td>
<td>v₃</td>
<td>ℓ₃</td>
</tr>
<tr>
<td>l₃</td>
<td>v₄</td>
<td>ℓ₄</td>
</tr>
<tr>
<td>s₀</td>
<td>v₅</td>
<td>ℓ₂</td>
</tr>
</tbody>
</table>

**Evaluation of** \( l₂ \cdot \text{min} \) **in** \( \langle \rho, \mu \rangle **:** \( \llbracket l₂ \cdot \text{min} \rrbracket \langle \rho, \mu \rangle **:** \( \llbracket l₂ \rrbracket \langle \rho, \mu \rangle = ℓ₃ **:**

\[ \llbracket l₂ \rrbracket \langle \rho, \mu \rangle = ℓ₃ \]
**Evaluation of expressions**

**MEMORY** $\mu$

<table>
<thead>
<tr>
<th>Event</th>
<th>lr</th>
<th>$o_1$</th>
<th>$l_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>hr</td>
<td>min</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>ListEvents</td>
<td>head</td>
<td>tail</td>
<td>null</td>
</tr>
<tr>
<td>10</td>
<td>15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Event</th>
<th>lr</th>
<th>$o_2$</th>
<th>$l_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListEvents</td>
<td>head</td>
<td>tail</td>
<td>null</td>
</tr>
<tr>
<td>18</td>
<td>20</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ENVIRONMENT** $\rho$

<table>
<thead>
<tr>
<th>Event</th>
<th>lr</th>
<th>$o_3$</th>
<th>$l_3$</th>
</tr>
</thead>
</table>

**EVALUATION OF** $l_2$.min in $\langle \rho, \mu \rangle$:

$[[l_2].min] \langle \rho, \mu \rangle$ = $l_3$

$\mu(l_3) = o_3$
EVALUATION OF EXPRESSIONS

MEMORY $\mu$ →

<table>
<thead>
<tr>
<th>Event</th>
<th>$hr$</th>
<th>$min$</th>
<th>$o_1$</th>
<th>$\ell_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListEvents</td>
<td>$head$</td>
<td>$tail$</td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>$10$</td>
<td>$15$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$20$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$\rho$ ENVIRONMENT

| $l_0$ | $v_1$ |
| $l_1$ |
| $l_2$ | $20$ |
| $l_3$ |
| $l_4$ |
| $s_0$ | $v_2$ |
| $v_3$ |
| $v_4$ |
| $v_5$ |

EVALUATION OF $l_2.min$ in $(\rho, \mu)$: $[[l_2.min]](\rho, \mu)$

$[[l_2]](\rho, \mu) = \ell_3$

$\mu(\ell_3) = o_3$

$[[l_2.min]](\rho, \mu) = o_3.min = 20$
**Alias Expressions**

We say that an expression $E \in \mathbb{E}$ is an alias expression of a variable $v \in V$ in a state $\sigma$ if and only if $⟦E⟧_\sigma = ⟦v⟧_\sigma$.

\[
\begin{align*}
\llbracket l_2.\text{min} \rrbracket (\rho, \mu) &= o_3.\text{min} = 20
\end{align*}
\]
**Alias Expressions**

We say that an expression $E \in E$ is an alias expression of a variable $v \in V$ in a state $\sigma$ if and only if $⟦E⟧_\sigma = ⟦v⟧_\sigma$.
We say that an expression $E \in \mathbb{E}$ is an alias expression of a variable $v \in \mathbb{V}$ in a state $\sigma$ if and only if $\llbracket E \rrbracket_\sigma = \llbracket v \rrbracket_\sigma$. 

$l_2.\text{min}$ is an alias expression of $l_1$ in $\langle \rho, \mu \rangle$. 

$$\llbracket l_2.\text{min} \rrbracket \langle \rho, \mu \rangle = o_3.\text{min} = 20$$ 

$$\llbracket l_1 \rrbracket \langle \rho, \mu \rangle = 20$$
**Abstract Interpretation Framework** [Cousot77]

Best correct approximation: \( f^{bca} = \alpha \circ f \circ \gamma \)

In practice: \( f^\# \) is less precise than \( f^{bca} \) and introduces loss of precision.
Concrete and Abstract Domains

- $\Sigma$ - set of all states
- $V = v_1, \ldots, v_{i+j}$ - set of all variables
- **Concrete Domain**: $C = \wp(\Sigma)$
- **Abstract Domain**: $A = (\wp(E))^{i+j}$
  - An abstract element $\langle A_1, \ldots, A_{i+j} \rangle$ contains, for each variable $v_r$, a set of expressions $A_r \subseteq E$ definitely aliased to $v_r$
  - Concrete states $\sigma$ corresponding to $\langle A_1, \ldots, A_{i+j} \rangle$ must satisfy the aliasing information represented by the latter, i.e., for each $v_r$, the value of all the expressions from $A_r$ in $\sigma$ must coincide with the value of $v_r$

**Concretization Map**:

$$\gamma(\langle A_1, \ldots, A_{i+j} \rangle) = \{ \sigma \in \Sigma \mid \forall v_r. \forall E \in A_r. \llbracket E \rrbracket \sigma = \llbracket v_r \rrbracket \sigma \}$$
**Constraint-based Static Analysis - Example**

- **Abstract Constraint Graph** \((ACG= \langle V, E \rangle)\) gives rise to an approximation of the aliasing information at each point of a program \(P\).

- The **CFG** of \(P\) gives rise to the nodes and arcs of the **ACG**, i.e., there is a node for every bytecode and there is an arc between 2 nodes if their corresponding bytecodes are adjacent in the CFG.

- Each node is decorated by an abstract element, i.e., by a tuple of sets of expressions representing a definite aliasing information at that point.

- Arcs propagate approximations of the aliasing information of their sources, i.e., they represent abstract semantics of bytecodes.
CONSTRAINT-BASED STATIC ANALYSIS - EXAMPLE

```
load 0 Event
getfield Event.min: int

load 1 int
add int
return int

catch
throw java.lang.Throwable
```
DEFINITE EXPRESSION ALIASING ANALYSIS

CONSTRAINT-BASED STATIC ANALYSIS - EXAMPLE

NODE c
- catch
- load 0 Event
- getfield Event.min: int
- catch
- throw java.lang.Throwable
- load 1 int
- add int
- return int

NODE a
- call Event.delayMinBy(int): int
- load 0 Event
- getfield Event.min: int
- catch
- throw java.lang.Throwable

NODE b
- store 3 int

NODE 1
- load 0 Event

NODE 2
- getfield Event.min: int

NODE 3
- load 1 int

NODE 4
- add int

NODE 5
- return int

NODE 6
- exit@delayMinBy

NODE 7
- catch
- load 1 int
- throw java.lang.Throwable

NODE 8
- add int

NODE 9
- exception@delayMinBy
CONSTRAINT-BASED STATIC ANALYSIS - EXAMPLE

Definite Expression Aliasing Analysis

Ahs abstract semantics

Constraint-based static analysis - example

load 0 Event
getfield Event.min: int

catch
throw java.lang.Throwable

load 1 int
add int
return int

catch
throw java.lang.Throwable

load 0 Event
getfield Event.min: int

load 1 int
add int
return int

catch
throw java.lang.Throwable

load 0 Event
getfield Event.min: int

load 1 int
add int
return int

catch
throw java.lang.Throwable

load 0 Event
getfield Event.min: int

load 1 int
add int
return int

catch
throw java.lang.Throwable

load 0 Event
getfield Event.min: int

load 1 int
add int
return int

catch
throw java.lang.Throwable

load 0 Event
getfield Event.min: int

load 1 int
add int
return int

catch
throw java.lang.Throwable
PROPAGATION RULES - EXAMPLE

INITIAL APPROXIMATION
\{v_3\}, \{2v_1\}, \{v_1\}

v_1 \quad v_2 \quad v_3

TYPE ENVIRONMENT

\begin{array}{|c|c|}
\hline
v_1 &= l_0 \\
E &= \text{Event} \\
\hline
v_2 &= l_1 \\
\text{min} &= \text{int} \\
\hline
v_3 &= s_0 \\
S &= \text{Event} \\
\hline
\end{array}

NODE 2
getfield Event.min:int

SEQUENTIAL ARC

\#5

NODE 5
load 1 int

Each variable different from the top of the stack keeps all its alias expressions in which the top of the stack does not appear. For each alias expression E of the top of the stack before getfield min, E.min becomes an alias expression of the top of the stack after getfield min if no evaluation of E might modify min.
Definite Expression Aliasing Analysis

Propagation Rules - Example

Initial Approximation
\{v_3\}, \{2v_1\}, \{v_1\}

Type Environment
\begin{array}{|c|c|}
\hline
L & v_1 = l_0, v_2 = l_1, v_3 = s_0 \\
\hline
S & \text{Event} \rightarrow \text{int} \\
\hline
\end{array}

NODE 2
getfield Event.min:int

Sequential Arc

NODE 5
load 1 int

Type Environment
\begin{array}{|c|c|c|}
\hline
L & v_1 = l_0, v_2 = l_1 \\
S & v_3 = s_0 \\
\hline
\text{Event} \rightarrow \text{int} \\
\hline
\end{array}
Propagation Rules - Example

Each variable different from the top of the stack keeps all its alias expressions in which the top of the stack does not appear.
**PROPAGATION RULES - EXAMPLE**

Each variable different from the top of the stack keeps all its alias expressions in which the top of the stack does not appear.

For each alias expression $E$ of the top of the stack before `getfield min`, $E$ becomes an alias expression of the top of the stack after `getfield min` if no evaluation of $E$ might modify $min$.
Propagating rules - example

Each variable different from the top of the stack keeps all its alias expressions in which the top of the stack does not appear.

For each alias expression $E$ of the top of the stack before `getfield min`, $E.min$ becomes an alias expression of the top of the stack after `getfield min` if no evaluation of $E$ might modify $min$. 
**Propagation Rules - Example**

- **Initial Approximation**
  - \( \{ v_3 \}, \{ 2v_1 \}, \{ v_1 \} \)
  - \( v_1, v_2, v_3 \)

- **Type Environment**
  - \( \begin{array}{c|c|c}
  L & S \\
  v_1 = l_0 & v_3 = s_0 \\
  v_2 = l_1 & \end{array} \)

- **Node 2**
  - `getfield Event.min:int`

- **Sequential Arc**
  - \( \#5 \)

- **Final Approximation**
  - \( \emptyset, \{ 2v_1 \}, \{ v_1.min \} \)
  - \( v_1, v_2, v_3 \)

- **Type Environment**
  - \( \begin{array}{c|c|c}
  L & S \\
  v_1 = l_0 & v_3 = s_0 \\
  v_2 = l_1 & \end{array} \)

- **Observations**
  - **Each variable different from the top of the stack keeps all its alias expressions in which the top of the stack does not appear.**
  - **For each alias expression \( E \) of the top of the stack before `getfield min`, \( E.min \) becomes an alias expression of the top of the stack after `getfield min` if no evaluation of \( E \) might modify \( min \).**
**PROPAGATION RULES - EXAMPLE**

Each variable different from the top of the stack keeps all its alias expressions in which the top of the stack does not appear.

For each alias expression $E$ of the top of the stack before `getfield min`, $E.min$ becomes an alias expression of the top of the stack after `getfield min` if no evaluation of $E$ might modify $min$.
PROPAGATION RULES - EXAMPLE

INITIAL APPROXIMATION
\{v_3\}, \{2v_1\}, \{v_1\}
\{\}
\{v_1\}
\{v_2\}
\{v_3\}

TYPE ENVIRONMENT
\begin{array}{|c|c|c|}
\hline
v_1 & v_2 & v_3 \\
\hline
l_0 & l_1 & s_0 \\
\hline
E v_1 & int & Event \\
\hline
\end{array}

NODE 2
getfield Event.min:int

EXCEPTIONAL ARC
#15

NODE 5
catch

\[ v_{\text{r}} < S = \{s_0\} \]
\[ A'_{\text{r}} = \{E \in A_{\text{r}} | v_3 \text{ does not appear in } E\} \]
\[ A'_{\text{r}} = \emptyset \]
**PROPAGATION RULES - EXAMPLE**

**INITIAL APPROXIMATION**

\[\{v_3\}, \{2v_1\}, \{v_1\}\]

**TYPE ENVIRONMENT**

- \(v_3 = s_0\)
- \(v_1 = l_0\)
- \(v_2 = l_1\)
- Event \(\rightarrow\) int \(\rightarrow\) Event

**NODE 2**

`getfield Event.min:int`

**EXCEPTIONAL ARC**

\(#15\)

**NODE 5**

`catch`

**TYPE ENVIRONMENT**

- \(v_3 = s_0\)
- \(v_1 = l_0\)
- \(v_2 = l_1\)
- Event \(\rightarrow\) int \(\rightarrow\) NPE
**PROPAGATION RULES - EXAMPLE**

**INITIAL APPROXIMATION**

\[ \{v_3\}, \emptyset, \{v_1\} \]

**NODE 2**

`getfield Event.min:int`

**TYPE ENVIRONMENT**

<table>
<thead>
<tr>
<th>(v_1)</th>
<th>(v_2)</th>
<th>(v_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(l_0)</td>
<td>(l_1)</td>
<td>(s_0)</td>
</tr>
<tr>
<td>Event</td>
<td>int</td>
<td>Event</td>
</tr>
</tbody>
</table>

**EXCEPTIONAL ARC**

\#15

**NODE 5**

`catch`

**TYPE ENVIRONMENT**

<table>
<thead>
<tr>
<th>(v_1)</th>
<th>(v_2)</th>
<th>(v_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(l_0)</td>
<td>(l_1)</td>
<td>(s_0)</td>
</tr>
<tr>
<td>Event</td>
<td>int</td>
<td>NPE</td>
</tr>
</tbody>
</table>

**IF** \(v_r \notin S = \{s_0 = v_3\}\),

\[ A'_r = \{E \in A_r \mid v_3 \text{ does not appear in } E\} \]
PROPAGATION RULES - EXAMPLE

IF $v_r \notin S = \{S_0 = v_3\}$,

$A'_r = \{E \in A_r \mid v_3 \text{ does not appear in } E\}$
PROPAGATION RULES - EXAMPLE

IF \( v_r \notin S = \{ S_0 = v_3 \} \),
\[ A'_r = \{ E \in A_r \mid v_3 \text{ does not appear in } E \} \]

IF \( v_r = v_3 \),
\[ A'_r = \emptyset \]
**PROPAGATION RULES - EXAMPLE**

### INITIAL APPROXIMATION

\[ \{v_3\}, \{2v_1\}, \{v_1\} \]

- \( v_1 \)
- \( v_2 \)
- \( v_3 \)

### TYPE ENVIRONMENT

\[ \begin{array}{|c|c|c|}
\hline
\text{Event} & \text{int} & \text{Event} \\
\hline
v_1 = l_0 & v_2 = l_1 & v_3 = s_0 \\
\hline
\end{array} \]

### NODE 2

**getfield Event.min:int**

### EXCEPTIONAL ARC

\#15

### FINAL APPROXIMATION

\[ \emptyset, \{2v_1\}, \emptyset \]

- \( v_1 \)
- \( v_2 \)
- \( v_3 \)

### TYPE ENVIRONMENT

\[ \begin{array}{|c|c|c|}
\hline
\text{Event} & \text{NPE} \\
\hline
v_1 = l_0 & v_2 = l_1 & v_3 = s_0 \\
\hline
\end{array} \]

### IF \( v_r \notin S = \{s_0 = v_3\} \),

\[ A'_r = \{E \in A_r \mid v_3 \text{ does not appear in } E\} \]

### IF \( v_r = v_3 \),

\[ A'_r = \emptyset \]
PROPAGATION RULES - EXAMPLE

IF $v_r \notin S = \{S_0 = V_3\}$,

$A'_r = \{E \in A_r \mid V_3 \text{ does not appear in } E\}$

IF $v_r = V_3$,

$A'_r = \emptyset$
DEFINITE EXPRESSION ALIASING ANALYSIS
has been implemented inside JULIA as a supporting analysis for
NULLNESS AND TERMINATION TOOLS
Effects of our Definite Expression Aliasing Analysis on the Nullness Analysis of Julia

- Precision improved by 45.98%
- Runtime increases by 9.88%

**Graph 1:**
- Precision: Nullness without aliasing analysis vs. Nullness with aliasing analysis
- Runtime: Nullness without aliasing analysis vs. Nullness with aliasing analysis

**Graph 2:**
- Precision: Runtime without aliasing analysis vs. Runtime with aliasing analysis
- Runtime: Runtime without aliasing analysis vs. Runtime with aliasing analysis
DEFINITE EXPRESSION ALIASING ANALYSIS ON THE TERMINATION ANALYSIS OF JULIA

PRECISION IMPROVED BY 11.44%  RUNTIME INCREASES BY 12.57%
**Goal:** define, formally prove correct and implement a **Definite Expression Aliasing Analysis for Java bytecode**

1. **Definition of a Concrete Operational Semantics of a Java bytecode-like Target Language**;
2. **Formal Definition of a Notion of Alias Expressions**;
3. **A Constraint-Based Inter-Procedural Static Analysis Based on Abstract Interpretation**;
4. **Formal Proof of Existence and Uniqueness of Solutions of Our Constraints**;
5. **Formal Proof of Correctness of the Analysis**;
6. **Implementation of Our Inter-Procedural Analysis for Full Java bytecode**;
7. **Experimental Evaluation of Our Approach on Real Life Benchmarks**.
Thank You!!!