Definite Expression Aliasing Analysis for Java Bytecode

Đurica Nikolić[1,2] and Fausto Spoto[1]

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

Bangalore, 25th September 2012

MOTIVATING EXAMPLE: HoneycombGallery

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

Motivating Example: HoneycombGallery

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

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

Motivating Example: HoneycombGallery

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

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

♦ IF THE TOP OF THE STACK IS NULL, GO TO ELSE

Motivating Example: HoneycombGallery

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

```
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

Motivating Example: HoneycombGallery

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

```
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

Motivating Example: HoneycombGallery

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

```
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

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

```
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

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?

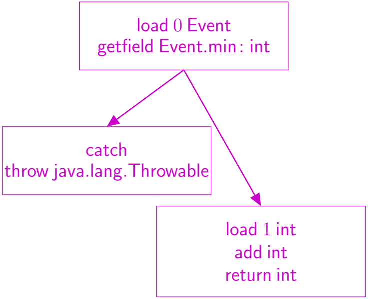
- expressions definitely aliased to `v` are non-null after `if (v!=null)`
- expressions `E.f` are non-null after an assignment `w.f=exp` if
  - `exp` is non-null and
  - `E` is definitely aliased to `w` and
  - evaluations of `E` must not update `f`
- inference of symbolic upper and lower bounds after comparison $x < y$:
  - expressions definitely aliased to `y` are upper bounds for `x`
  - expressions definitely aliased to `x` are lower bounds for `y`

Target language: Java bytecode

```
public class Event {
  public int hr, min;
  ...
  public int delayMinBy(int offset) {
    return min + offset;
  }
  ...
}
```

Target language: Java bytecode

```
public class Event {
  public int hr, min;
  ...
  public int delayMinBy(int offset) {
    return min + offset;
  }
  ...
}
```



STATE

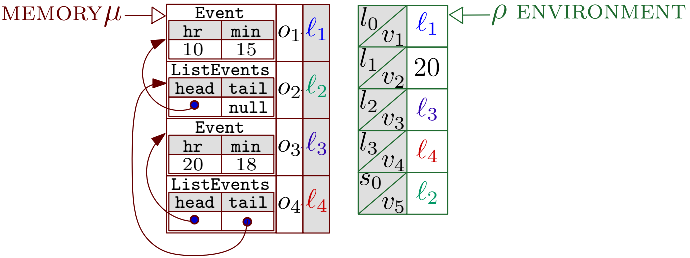
LOCAL VARIABLES: $L = \{l_0, \ldots, l_{i-1}\}$ STACK ELEMENTS: $S = \{s_0, \ldots, s_{j-1}\}$

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

State

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

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



$\sigma = \langle\langle[@\ell_1, 20, @\ell_3, @\ell_4] \parallel @\ell_2\rangle, \mu\rangle$

Expressions

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

Expressions

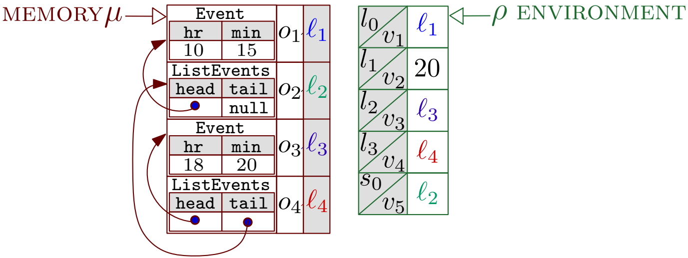
- $\mathcal{F}$ - set of all field names
- $\mathcal{M}$ - set of all method names

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

$$
\begin{array}{rcll}
\mathbb{E} \ni E & ::= & n & \text{constants} \\
& | & v & \text{variables} \\
& | & E \oplus E & \text{arithmetic expressions} \\
& | & E.f & \text{field accesses} \\
& | & E.m(E, \ldots) & \text{method invocations,}
\end{array}
$$

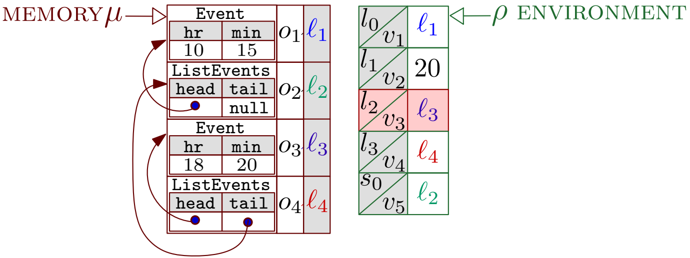
where $n \in \mathbb{Z}$, $v \in V$, $\oplus \in \{+, -, \times, \div, \%\}$, $f \in \mathcal{F}$ and $m \in \mathcal{M}$.

EVALUATION OF EXPRESSIONS



EVALUATION OF l_2.min IN $\langle\rho,\mu\rangle$: $[\![l_2.\mathtt{min}]\!]\langle\rho,\mu\rangle$

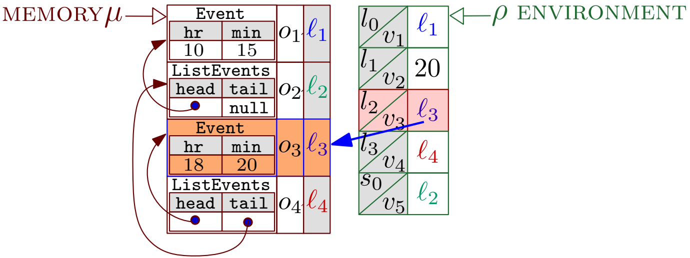
EVALUATION OF EXPRESSIONS



EVALUATION OF $l_2.\mathtt{min}$ IN $\langle\rho,\mu\rangle$: $[\![l_2.\mathtt{min}]\!]\langle\rho,\mu\rangle$

$[\![l_2]\!]\langle\rho,\mu\rangle = \ell_3$

Evaluation of expressions

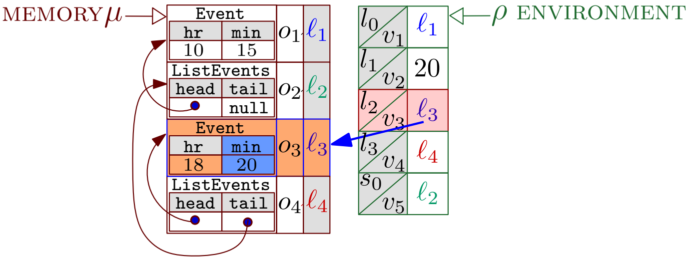


Evaluation of l_2.min in $\langle\rho,\mu\rangle$: $[\![l_2.\texttt{min}]\!]\langle\rho,\mu\rangle$

$[\![l_2]\!]\langle\rho,\mu\rangle = \ell_3$

$\mu(\ell_3) = o_3$

Evaluation of expressions



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

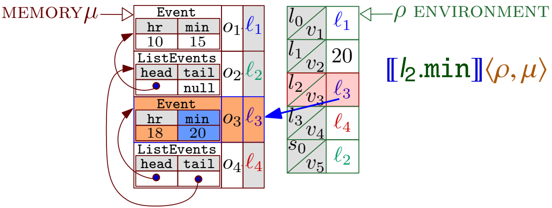
$[\![l_2]\!]\langle\rho,\mu\rangle = \ell_3$

$\mu(\ell_3) = o_3$

$[\![l_2.\mathtt{min}]\!]\langle\rho,\mu\rangle = o_3.\mathtt{min} = 20$

Alias Expressions

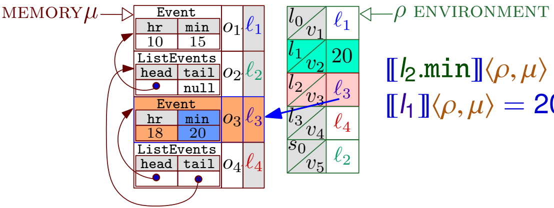
We say that an expression $E \in \mathbb{E}$ is an **alias expression** of a variable $v \in V$ in a state σ if and only if $[\![E]\!]\sigma = [\![v]\!]\sigma$.



$$[\![l_2.\mathtt{min}]\!]\langle\rho,\mu\rangle = o_3.\mathtt{min} = 20$$

Alias Expressions

We say that an expression $E \in \mathbb{E}$ is an alias expression of a variable $v \in V$ in a state σ if and only if $[\![E]\!]\sigma = [\![v]\!]\sigma$.

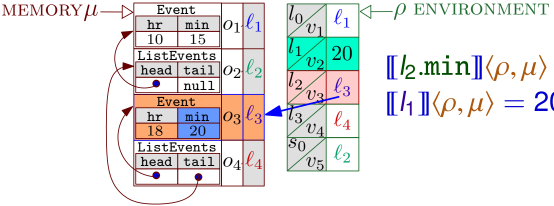


$[\![l_2.\mathtt{min}]\!]\langle\rho,\mu\rangle = o_3.\mathtt{min} = 20$

$[\![l_1]\!]\langle\rho,\mu\rangle = 20$

Alias Expressions

We say that an expression $E \in \mathbb{E}$ is an alias expression of a variable $v \in V$ in a state σ if and only if $[\![E]\!]\sigma = [\![v]\!]\sigma$.

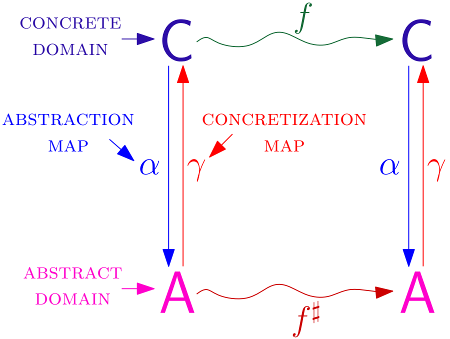


$[\![l_2.\mathtt{min}]\!]\langle\rho,\mu\rangle = o_3.\mathtt{min} = 20$

$[\![l_1]\!]\langle\rho,\mu\rangle = 20$

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

Abstract Interpretation Framework [CousotCousot77]



BEST CORRECT APPROXIMATION: $f^{bca} = \alpha \circ f \circ \gamma$

IN PRACTICE: $f^{\sharp}$ IS LESS PRECISE THAN f^{bca} AND INTRODUCES LOSS OF PRECISION

Concrete and Abstract Domains

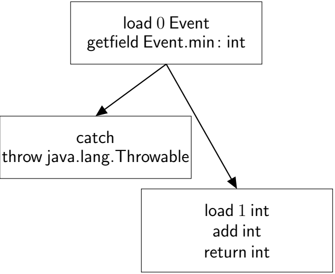
- Σ - set of all states
- $V = v_1, \ldots, v_{i+j}$ - set of all variables
- Concrete Domain: $C = \wp(\Sigma)$
- Abstract Domain: $A = (\wp(\mathbb{E}))^{i+j}$
  - an abstract element $\langle A_1, \ldots, A_{i+j} \rangle$ contains, for each variable v_r, a set of expressions $A_r \subseteq \mathbb{E}$ definitely aliased to v_r
  - concrete states σ 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 σ must coincide with the value of v_r
- Concretization Map:

$$\gamma(\langle A_1, \ldots, A_{i+j} \rangle) = \{\sigma \in \Sigma \mid \forall v_r.\forall E \in A_r.[\![E]\!]\sigma = [\![v_r]\!]\sigma\}$$

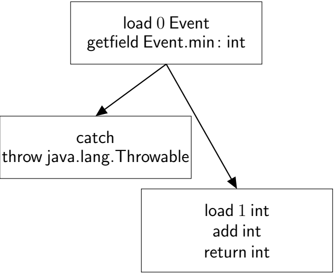
Constraint-based static analysis - example

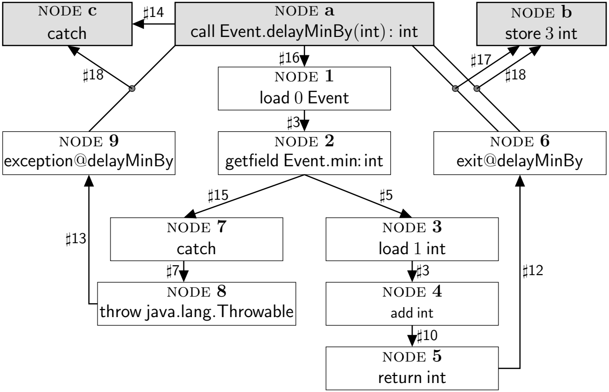
- Abstract Constraint Graph (ACG= $\langle V, E \rangle$) gives rise to an approximation of the aliasing information at each point of a program P.
- The CFG of P gives rise to the nodes and arcs of the ACG, i.e., there is a node for every bytecode and there is an arc between 2 nodes if their corresponding 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

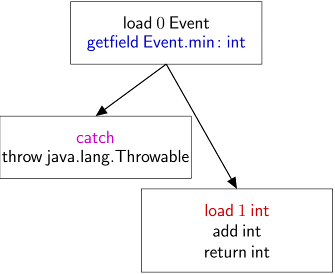


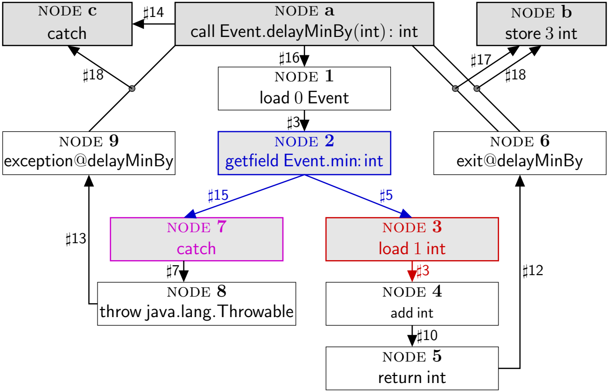
Constraint-based static analysis - example



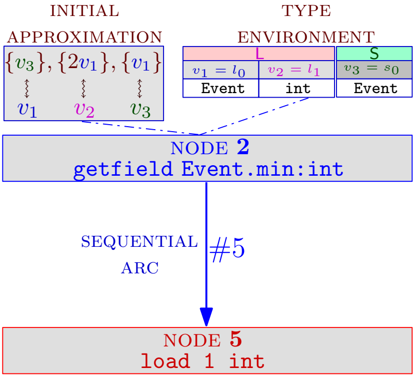


Constraint-based static analysis - example

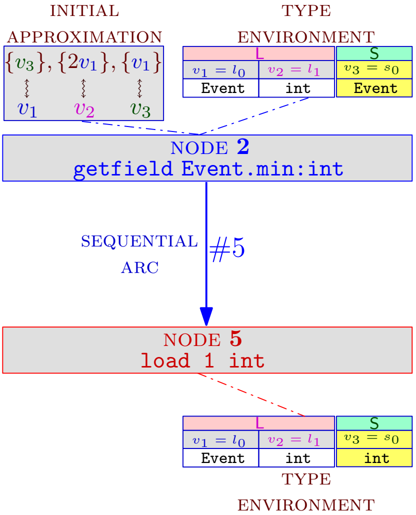




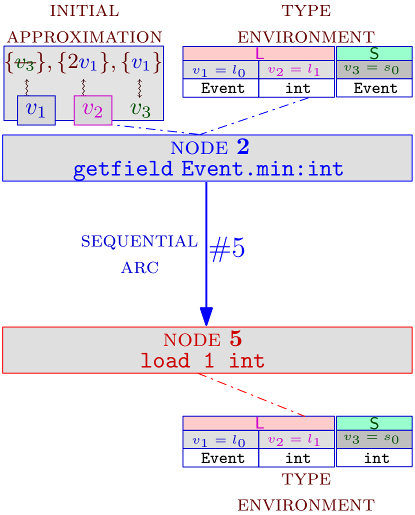
Propagation rules - example



Propagation rules - example

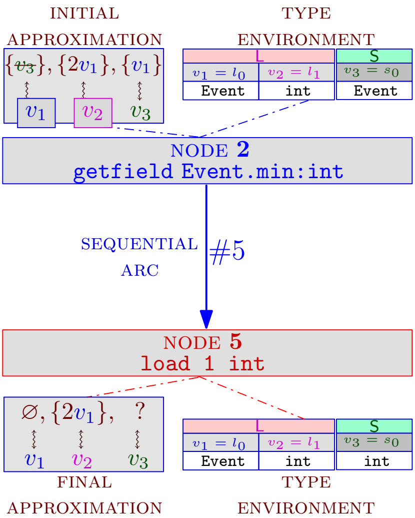


Propagation rules - example



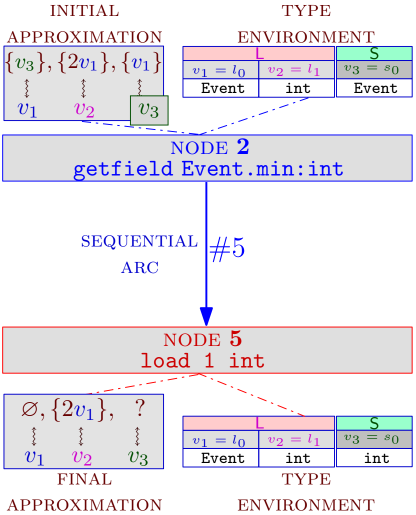
- Each variable different from the top of the stack keeps all its alias expressions in which the top of the stack does not appear

Propagation rules - example



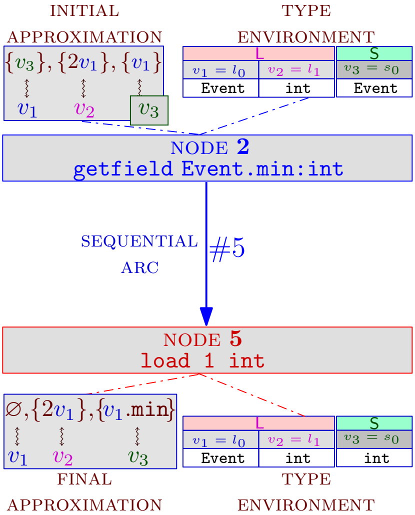
- Each variable different from the top of the stack keeps all its alias expressions in which the top of the stack does not appear

Propagation rules - example



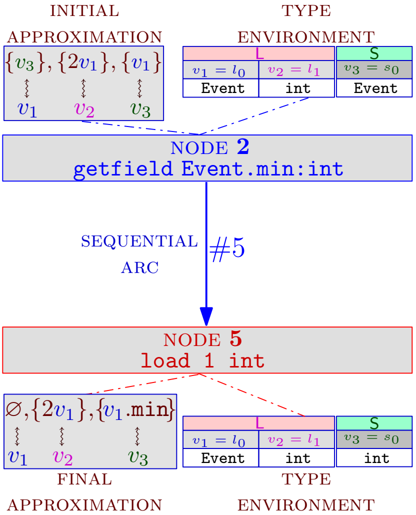
- Each variable different from the top of the stack keeps all its alias expressions in which the top of the stack does not appear
- For each alias expression E of the top of the stack before `getfield min`, E.`min` becomes an alias expression of the top of the stack after `getfield min` if no evaluation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

Propagation rules - example



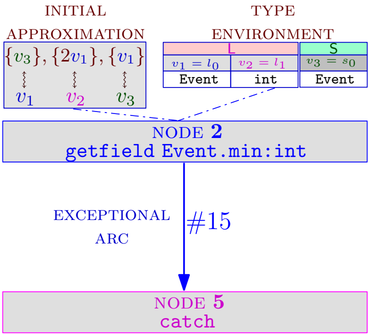
- Each variable different from the top of the stack keeps all its alias expressions in which the top of the stack does not appear
- For each alias expression E of the top of the stack before `getfield` `min`, E.`min` becomes an alias expression of the top of the stack after `getfield` `min` if no evaluation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

Propagation rules - example

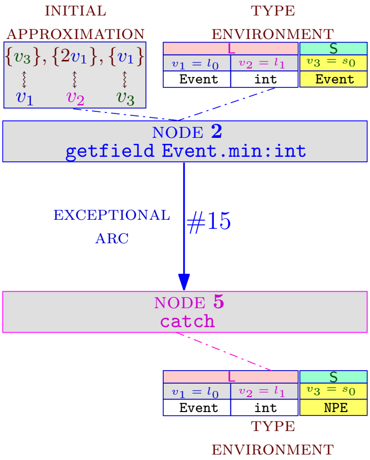


- Each variable different from the top of the stack keeps all its alias expressions in which the top of the stack does not appear
- For each alias expression E of the top of the stack before `getfield min`, E.min becomes an alias expression of the top of the stack after `getfield min` if no evaluation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

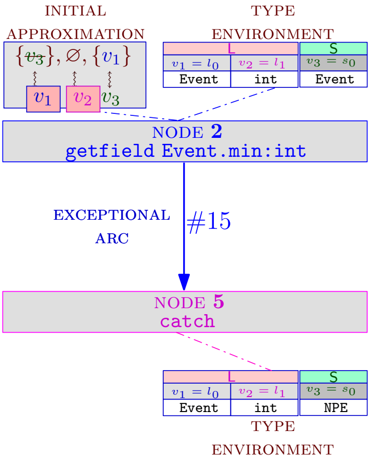
Propagation rules - example



PROPAGATION RULES - EXAMPLE

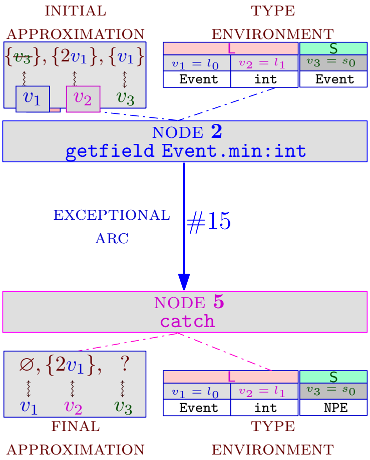


PROPAGATION RULES - EXAMPLE



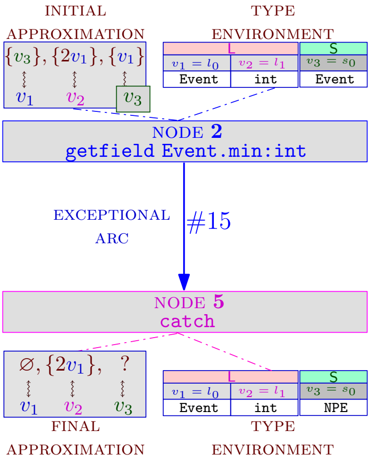
- IF $v_r \notin S = \{s_0 = v_3\}$,
  $A'_r = \{E \in A_r \mid v_3$ DOES NOT APPEAR IN $E\}$

Propagation rules - example



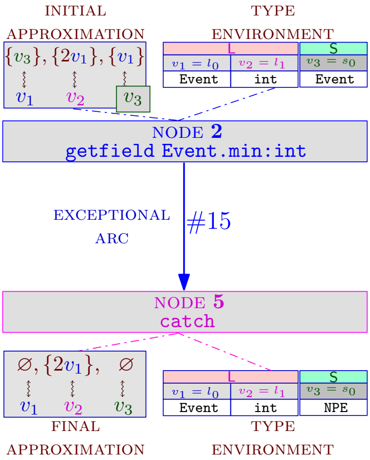
- IF $v_r \notin S = \{s_0 = v_3\}$,
  $A'_r = \{E \in A_r \mid v_3 \text{ DOES NOT APPEAR IN } E\}$

Propagation rules - example



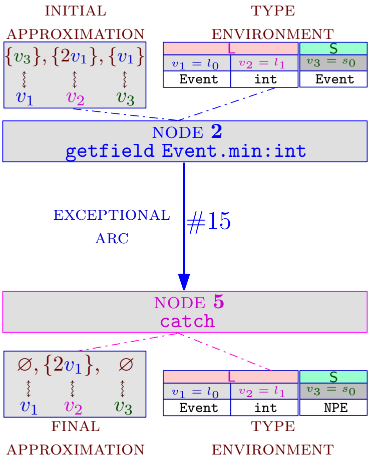
- IF $v_r \notin S = \{s_0 = v_3\}$,
  $A'_r = \{E \in A_r \mid v_3$ DOES NOT APPEAR IN $E\}$
- IF $v_r = v_3$,
  $A'_r = \varnothing$

Propagation rules - example



- IF $v_r \notin S = \{s_0 = v_3\}$,
  $A'_r = \{E \in A_r \mid v_3$ DOES NOT APPEAR IN $E\}$
- IF $v_r = v_3$,
  $A'_r = \varnothing$

Propagation rules - example



- IF $v_r \notin S = \{s_0 = v_3\}$,
  $A'_r = \{E \in A_r \mid v_3$ DOES NOT APPEAR IN $E\}$
- IF $v_r = v_3$,
  $A'_r = \varnothing$

Julia - a static analyzer for Java and Android

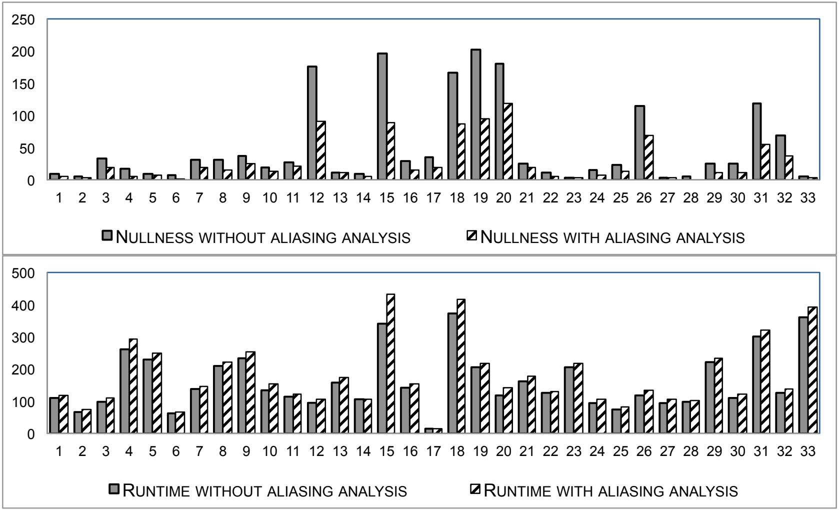
Definite Expression Aliasing Analysis

has been implemented inside Julia as a supporting analysis for

Nullness and Termination tools

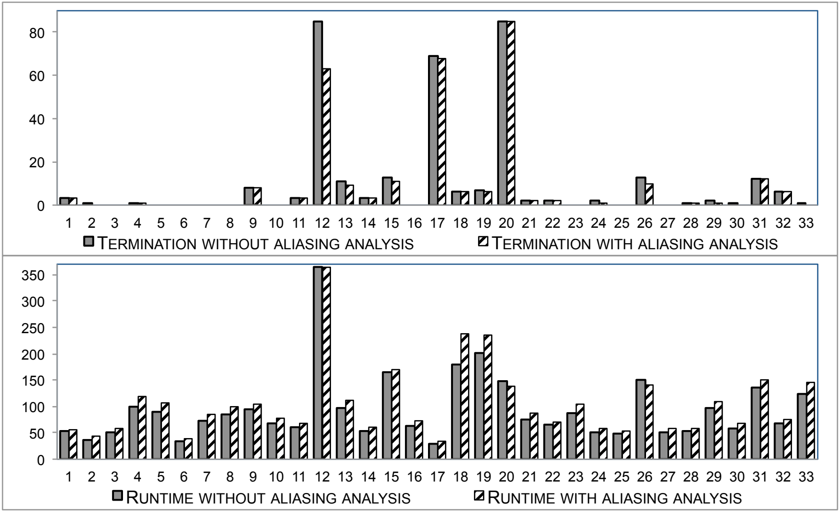
Effects of our Definite Expression Aliasing Analysis on the Nullness Analysis of Julia

PRECISION IMPROVED BY 45.98% **RUNTIME INCREASES BY 9.88%**



Effects of our 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!!!