Interval Data Flow Analysis

By: Jake Zych

Project Goal:

Detect out of bounds array accesses in Java using Interval Data Flow Analysis

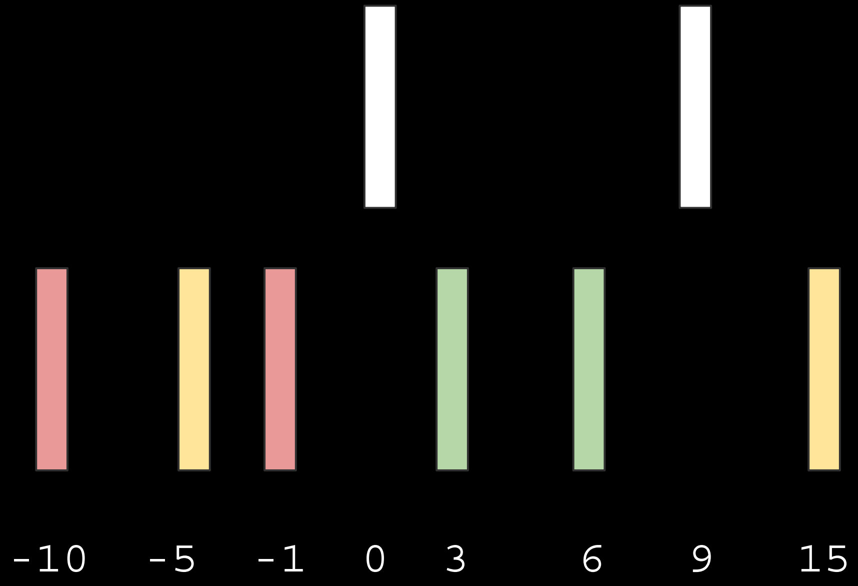
Background:

$$
\begin{array}{rcl}
L & = & \mathbb{Z}_\infty \times \mathbb{Z}_\infty \qquad \text{where } \mathbb{Z}_\infty = \mathbb{Z} \cup \{-\infty, \infty\} \\
[l_1, h_1] \sqsubseteq [l_2, h_2] & \textit{iff} & l_2 \leqslant_\infty l_1 \wedge h_1 \leqslant_\infty h_2 \\
[l_1, h_1] \sqcup [l_2, h_2] & = & [min_\infty(l_1, l_2), max_\infty(h_1, h_2)] \\
\top & = & [-\infty, \infty] \\
\bot & = & [\infty, -\infty] \\
\sigma_0 & = & \top \\
\alpha(x) & = & [x, x]
\end{array}
$$

$$
\begin{array}{lll}
f_I[\![x := y + z]\!](\sigma) & = \sigma[x \mapsto [l, h]] & \text{where } l = \sigma(y).low +_\infty \sigma(z).low \\
 & & \text{and } h = \sigma(y).high +_\infty \sigma(z).high \\
f_I[\![x := y + z]\!](\sigma) & = \sigma & \text{where } \sigma(y) = \bot \vee \sigma(z) = \bot
\end{array}
$$

Unsafe Array Access Detection

```
int[] r = new int[10]
...
x -> [-5, 15]  // WARNING
y -> [3, 6]    // OKAY
z -> [-10, -1] // ERROR
```



Why Interval Analysis?

Sign Analysis

- No numeric information
- Less precise results
- Does not map to problem space as well

Constant Propagation:

- Loses precision more easily
- If `x` can map to both `c1` and `c2`
  - Constant Propagation:
    - `x -> Top`
  - Interval Analysis:
    - `x -> [c1, c2] (c1 < c2)`

Technologies

Preliminary Implementation Results

Identifies index out of bounds errors successfully for programs containing:

- Function Calls
- Branches
- Arithmetic Operations

```java
public static void testWhile() {
    int x, y;
    int[] array = new int[5];
    y = -10;
    x = 0;
    while (x != y) {
        x = x + 1;
    }
    y = 0;
    int ignore = array[y]; // OKAY
    ignore = array[x]; // WARNING
}
```

```
<=========----> 72% EXECUTING [1m 49s]
> :test > 0 tests completed
> :test > Executing test range_analysis.WhileLoopTest
```

Not Terminating?

Widening Operator

$$
\begin{aligned}
W(\perp, l_{current}) &= l_{current} \\
W([l_1, h_1], [l_2, h_2]) &= [min_W(l_1, l_2), max_W(h_1, h_2)] \\
&\quad \text{where } min_W(l_1, l_2) = l_1 \qquad \text{if } l_1 \leqslant l_2 \\
&\quad \text{and } min_W(l_1, l_2) = -\infty \qquad \text{otherwise} \\
&\quad \text{where } max_W(h_1, h_2) = h_1 \qquad \text{if } h_1 \geqslant h_2 \\
&\quad \text{and } max_W(h_1, h_2) = \infty \qquad \text{otherwise}
\end{aligned}
$$

Without Widening

```
Chain: Bottom,[0, 0],
[0, 1], [0, 2], ...
```

With Widening

```
Chain: Bottom,[0, 0],
[0, ∞], [0, ∞], ...
```

- ➩ Applied before processing the head of a loop
- ➩ Can limit precision of the analysis
- ➩ Soot framework does not automatically differentiate between if statements and loop guards

Constant Prediction

$$
\begin{aligned}
W(\perp, l_{current}) &= l_{current} \\
W([l_1,h_1],[l_2,h_2]) &= [min_W(l_1,l_2), max_W(h_1,h_2)] \\
&\quad \text{where } min_W(l_1,l_2) = l_1 && \text{if } l_1 \leqslant l_2 \\
&\quad \text{and } min_W(l_1,l_2) = \boxed{-\infty} && \text{otherwise} \\
&\quad \text{where } max_W(h_1,h_2) = h_1 && \text{if } h_1 \geqslant h_2 \\
&\quad \text{and } max_W(h_1,h_2) = \boxed{\infty} && \text{otherwise}
\end{aligned}
$$

K -> Set of All Constants in the Program

With Constant Prediction

Chain: Bottom, [0, 0], [0, k], [0, k], ...

OR

Chain: Bottom, [0, 0], [0, k], [0, ∞], [0, ∞], ...

$$
\begin{aligned}
W(\perp, l_{current}) &= l_{current} \\
W([l_1,h_1],[l_2,h_2]) &= [min_K(l_1,l_2), max_K(h_1,h_2)] \\
&\quad \text{where } min_K(l_1,l_2) = l_1 && \text{if } l_1 \leqslant l_2 \\
&\quad \text{and } min_K(l_1,l_2) = \boxed{max(\{k \in K \mid k \leqslant l_2\})} && \text{otherwise} \\
&\quad \text{where } max_K(h_1,h_2) = h_1 && \text{if } h_1 \geqslant h_2 \\
&\quad \text{and } max_K(h_1,h_2) = \boxed{min(\{k \in K \mid k \geqslant h_2\}} && \text{otherwise}
\end{aligned}
$$

Widening Operator Results

```java
// Tests the widening operator making use of constants in the program to bound values
public static void testConstantTruncation() {
    int x, y;
    int[] array = new int[10];
    x = 1;
    y = 1;
    while (x != 10) {
        x = x + 1;
        y = y - 1;
    }

    int ignore = array[x - 1]; // OKAY x - 1 = [9, 9]
    ignore = array[y]; // WARNING y = [-inf, 1]
}
```

```
Calling Widen
Previous Sigma: {i1=[1, 1], i2=[1, 1], i4=[-2147
Current Sigma: {i1=[1, 2], i2=[0, 1], i4=[-21474
maxWiden val: 10
minWiden val: -2147483648
Resulting Sigma{i1=[1, 10], i2=[-2147483648, 1],
```

```
i1 = x
i2 = y
```

Example Program From: Program Analysis (Aldrich, Le Goues, Padhye)

Testing on Real World Software

```java
// Max-Heap Function
public static void maxheap(int[] a, int i){
    left=2*i;
    right=2*i+1;
    System.out.println(i + " " + left + " " + right);
    if(left <= n && a[left] > a[i]){
            largest=left;
    }
    else{
            largest=i;
    }

    if(right <= n && a[right] > a[largest]){
            largest=right;
    }
    if(largest!=i){
            exchange(i,largest);
            maxheap(a, largest);
    }
}
```

Original Heap Sort Implementation
https://github.com/farhankhwaja/HeapSort/blob/master/HeapSort.java

```java
public static int maxheap(int[] a, int i, int n){
    int left, right;
    left=2*i;
    right=2*i+1;
    int largest;
    // instrumented for reportWarnings
    int[] ignore = new int[5];
    int x = ignore[left]; // OKAY
    x = ignore[i]; // OKAY
    x = ignore[right]; // OKAY

    if(left <= n && a[left] > a[i]){
        largest =left;
    }
    else{
        largest =i;
    }

    if(right <= n && a[right] > a[largest]){
        largest =right;
    }
    if(largest !=i){
        exchange(a, i, largest);
        maxheap(a, largest, n);
    }
    x = ignore[largest]; // OKAY
    return 0;
```

Instrumented Code used to Test

Testing on Real World Software

```java
// Exchange Function
public static void exchange(int i, int j){
    int t=a[i];
    a[i]=a[j];
    a[j]=t;
    }
```

Original Heap Sort Implementation
https://github.com/farhankhwaja/HeapSort/blob/master/HeapSort.java

```java
// Exchange Function
public static int exchange(int a[], int i, int j){
    int []ignore = new int[5]; // instrumented for reportWarnings
    int t=a[i];
    a[i]=a[j];
    a[j]=t;
    // instrumented for reportWarnings
    int x = ignore[i]; // OKAY
    int y = ignore[j]; // OKAY
    return 0;
}
```

Instrumented Code used to Test

Other Real World Software Testing

```java
    /**
     * Assumes a < b.
     */
    private double arcLengthRecursive(int indexA, double remainderA, int indexB, double remainderB) {
        switch (indexB - indexA) {
            case 0:
                return arcLengthRecursive(indexA, remainderA, remainderB);

            case 1:
                // This case is merely a speed-up for a very common case
                return arcLengthRecursive(indexA, remainderA, 1.0)
                    + arcLengthRecursive(indexB, 0.0, remainderB);

            default:
                return arcLengthRecursive(indexA, remainderA, indexB - 1, 1.0)
                    + arcLengthRecursive(indexB, 0.0, remainderB);
        }
    }

    private double arcLengthRecursive(int index, double remainderA, double remainderB) {
        final Vector3 position1 = nodes.get(index).getPosition();
        final Vector3 position2 = nodes.get(index + 1).getPosition();

        return position1.distance(position2) * (remainderB - remainderA);
    }
```

Linear Interpolation from WorldEdit, a popular [MineCraft Mod]

➪ Less Successful Analysis

Interval Analysis as a Bug Catching Tool

- Requires instrumenting target code for test, in addition to potentially the analysis code itself
  - Low extensibility
- Fuzzing/Testing may be preferred when searching for indexing errors
  - May find other bugs as well
- However, Interval Analysis gives the exact location of the bug
- Tradeoff between performance and precision
  - Illustrated by call string context

References

- https://cmu-program-analysis.github.io/2022/resources/program-analysis.pdf
- https://github.com/farhankhwaja/HeapSort/blob/master/HeapSort.java
- https://github.com/EngineHub/WorldEdit/blob/master/worldedit-core/src/main/java/com/sk89q/worldedit/math/interpolation/LinearInterpolation.java
- https://www.sable.mcgill.ca/soot/doc/index.html
- https://github.com/jakezych/range-analysis

Questions?