SOFTWARESYNTHESIS 236347

Project 2: Static Buffer Overrun Analysis

14/3/2013

Hagai Attias 036860682
Maayan Arbiv 037117132
Ari Zigler 036853141

General description

We have implemented a static analyzer based on *Soot* infrastructure, used to detect and warn about potential illegal array accesses (Index out of bounds). This tool implements intraprocedural analysis and analyzes .java files only.

In order to do so, we have implemented an interval analysis (as seen in class), where we keep for each program point the set of locals defined before this point, with their corresponding interval.
This analysis is a *May* problem, and a *Forward flow* direction.

Our analysis supports the following operators (invoked on intervals) : op+, op-, op*, op/, neg, pair<, pair<=, pair== , pair!= . For any other operator we do not currently support, the resulting interval(s) will get $[-\infty,\infty]$ in order to be conservative and prevent false negatives.

To implement the boolean operators, we use Soot's `ForwardBranchedFlowAnalysis` Interface to enforce the condition on the intervals differently for the branch route (where condition holds) and the fall-through route (where condition doesn't hold).

The analysis phase is as follows:

Pass #1: Invoke *Interval* analysis to compute the set of intervals for each program point.

Pass #2: Invoke *Array Definitio*n analysis to compute for each program point, the set of arrays defined before this point, with their corresponding size interval, determined from Pass #1.

Pass #3: Invoke Array Bounds Check analysis to check for each array access in the program, whether this is a possible illegal access, based on the array size interval from Pass #2 and index size interval from Pass #1.

Note that generally it is possible to unite the 3 passes to actual one pass. However, the only pass that perform actual computation is Pass #1 for the interval analysis. In order to isolate the interval analysis and keep it generic so it can be used for other uses, and for simplicity, we divided the analysis to these 3 passes.

Our analysis is capable to distinguish between a definite illegal access to a potential one.
In order to do so, we compute two possible array sizes based on the array size interval $[a,b]$ (For simplicity assume this $[a,b]$ interval is > 0): Minimum size interval is $[0,a-1]$ and Maximum size interval is $[0,b-1]$.
Assume the index access interval is $[c,d]$. If the maximum size interval $[0,b-1]$ does not intersect with $[c,d]$, we mark this access as a definite illegal access. Otherwise, we check if this access is possibly illegal according to the minimum array size interval (to keep conservative), and if so, we mark it as potential illegal access. We also notify whether the access is illegal because of a lower bound (array[-1]) or because of an upper bound.

We apply delayed widening in order to improve the precision of the analysis. For performance issues, we set our max iterations constant to 50. After visiting a node in the CFG more than 50 times, we set its interval to the extreme, only in the direction that changes,
e.g. the statement x = x + 1 in a loop will only change x upper bound to ∞.

Our output is displayed to the user using the *Soot* plugin, displaying an ***SA*** mark for each illegal access line, along with the information message displayed as a tooltip.

Formal description of the transformers

Abstract domain: Intervals Domain as described in lectures.

Transformers:

We use F^i as defined in lecture 6, where $action_i$ defined as follows:

Assignment:

$$[x := a]_i(m) = m[x \to v] \; where < a, m > \Downarrow_i v$$

Above, a is an expression which in big step semantics evaluates to $v \in L^i$ using mathematical operators.

Also, we use the following as defined in lectures:

$$[a,b] \sqcap_i [c,d] = [meet(max(\{a,c\}), min(\{b,d\}))]$$

$$meet(a,b) = \begin{cases} [a,b] & a \le b \\ \bot & \text{otherwise} \end{cases}$$

$$[a,b] \sqcup_i [c,d] = [\min(a,c), \max(b,d)]$$

Arithmetic operators in our abstract domain are defined as:

$MathOp: L^i \to L^i$ Where$MathOp \in \{+, -, *, /, neg\}$:

$$Op+([a,b],[c,d]) = ([a+b, c+d])$$

$$Op-([a,b],[c,d]) = ([a-b, c-d])$$

$$Op*([a,b],[c,d]) = ([\min(a*c, a*d, b*c, b*d), \max(a*c, a*d, b*c, b*d)])$$

$$neg([a,b]) = ([-b,-a])$$

$$Op/([a,b],[c,d]) = \begin{cases} ([-\infty,\infty]) & c = d = 0 \\ ([-\max(|a|,|b|), \max(|a|,|b|)]) & c < 0, d > 0 \\ ([\min(a/1, a/d, b/1, b/d), \max(a/1, a/d, b/1, b/d)]) & c = 0, d > 0 \\ ([\min(a/c, -a/1, b/c, -b/1), \max(a/c, -a/1, b/c, -b/1)]) & c < 0, d = 0 \\ ([\min(a/c, a/d, b/c, b/d), \max(a/c, a/d, b/c, b/d)]) & otherwise \end{cases}$$

Boolean Operators:

$$Pair \le ([a,b],[c,d]) = ([a,b] \sqcap_i [-\infty, d], [a,\infty] \sqcap_i [c,d])$$

$$Pair < ([a,b],[c,d]) = ([a,b] \sqcap_i [-\infty, d-1], [a,\infty] \sqcap_i [c,d])$$

$$Pair == ([a,b],[c,d]) = ([a,b] \sqcap_i [c,d], [a,b] \sqcap_i [c,d])$$