Using Range Analysis for Software Verification

Aleksandr Zaks

SVV’06 - International Workshop on Software Verification and Validation

Joint work with:

Ilya Shlyakter, Franjo Ivančić, Hari Cadambi, Zijiang Yang, Malay Ganai, Aarti Gupta, and Pranav Ashar
Our Goal and Suggested Approach

- Reduce possible state space of a program to speed up later stages of verification. In particular, we are interested in improving *model checking*.

- Our main approach is based on generating a linear constraint system, which solution represents possible value ranges for program variables. The framework is an extension of the work by Radu Rugina and Martin Rinard described in "Symbolic bounds analysis of pointers, array indices, and accessed memory regions".

- Our second approach, is tailored specifically for *bounded model checking*, and relies on the fact that we are only interested in traces of length up to $k$. 
More About Bounds

- At each program location, we derive lower and upper bounds for each local integer variable.

- Bounds are linear combinations of symbolic variables, which represent initial values of parameters of an enclosing procedure.

- An example of a bound expression for a function $\textit{foo}(\text{int } n)$ is $L_{\text{pre}_3}^v = 3 \times n_0 + 5$, where $v$ is a local integer variable declared in $\textit{foo}$.

- Whenever we make a call $\textit{foo}(2)$, we can safely assume $v \geq 11$. 
int foo(int l) {
    int t = l + 2;
    if (t > 6)
        t -= 3;
    else
        t--;
    return t;
}

void bar() {
    int x = 3, y = x - 3;
    while (x <= 4) {
        y++;
        x = foo(x);
    }
    y = foo(y);
}
Intra-procedural Bounds Analysis

- We only consider intra-procedural analysis. However, due to FSOFT’s modeling that is not crucial, since all the function calls are "inlined".

- Essentially, we have implemented a so called imprecise inter-procedural analysis where we consider invalid paths.

- The approach can be extended to be "truly" inter-procedural, similar to the original framework by Radu Rugina and Martin Rinard.
Some Preliminaries

- $V$ be a set of all local integer variables.
- $P$ be a set of all integer parameters passed by value.
- $L$ be a set of all locations. Each basic block $B_i$ defines two locations:
  - $\text{pre}_i$ the beginning of the block $B_i$
  - $\text{post}_i$ the end of the block $B_i$.
- We assume $B_0$ is the initial block and $\text{pre}_0$ is the entry point.
Bounds as Linear Expressions with Unknown Coefficients

• We are trying to find a solution for $L^v_{loc}$ and $U^v_{loc}$ in the following form:

$$L^v_{loc} = C^L + \sum_{p \in P} C_p^L \cdot p_0,$$

$$U^v_{loc} = C^U + \sum_{p \in P} C_p^U \cdot p_0.$$

• $C^i$s are the unknown rational coefficients we are looking for.
Initialization Constraints

- For each $p \in P$
  \[
  \begin{align*}
  L_{p_{pre_0}}^p &= p_0 \\
  U_{p_{pre_0}}^p &= p_0
  \end{align*}
  \]

- For each $v \in V \setminus P$
  \[
  \begin{align*}
  L_{v_{pre_0}}^v &= -\infty \\
  U_{v_{pre_0}}^v &= +\infty
  \end{align*}
  \]

- Note that for unsigned int $-\infty$ corresponds to 0
Assignment Constraints

- For each assignment of the form $v = e$ in a basic block $B_i$, such that $v \in V$, we add the following constraints:

$$L^v_{post_i} = l(e, pre_i)$$
$$U^v_{post_i} = u(e, pre_i),$$

where $l/u(e, loc)$ represents a lower/upper bound of the $e$ at location $loc$.

- Formally, $l(e, loc)$ is defined as:

$$l(c, loc) = c$$
$$l(v, loc) = L^v_{loc}, \text{ if } v \in V$$
$$l(e_1 \pm e_2, loc) = l(e_1, loc) \pm l(e_2, loc)$$

$$l(c \ast e) = \begin{cases} 
    c \ast l(e, loc) & \text{if } c \geq 0 \\
    c \ast u(e, loc) & \text{if } c < 0
\end{cases}$$

$$l(e, loc) = -\infty, \text{ in all other cases}$$

- Similarly, we define $u(e, loc)$ for upper symbolic bounds.
Flow Constraints

- Each transition \((B_i, B_j)\) adds the following flow constraints for each variable \(v \in V\):

\[
\begin{align*}
L^v_{pre_j} & \leq L^v_{post_i} \\
U^v_{pre_j} & \geq U^v_{post_i}.
\end{align*}
\]

- At this point our system is compressive enough to guarantee soundness of the bounds. To remind, we have the following types of constraints so far:
  - initialization,
  - assignment,
  - flow.

- However, we can significantly increase precision by analyzing guards.
Using Conditionals

- Whenever transition \((B_i, B_j)\) is guarded by an expression of the form \(v \geq e\), we can relax the flow constraint for the lower bound, \(L^v_{pre_j} \leq L^v_{post_i}\), as follows:

\[
L^v_{pre_j} \leq L^v_{post_i} \lor L^v_{pre_j} \leq l(e, post_i).
\]

- Note that if we can take the transition \((B_i, B_j)\), \(v \in [l(e, post_i), +\infty)\) and \(L^v_{post_i} = l(e, post_i)\), then

\[
L^v_{pre_j} \leq l(e, post_i) \rightarrow L^v_{pre_j} \leq L^v_{post_i}.
\]

and the soundness of the ranges is preserved.

- Since we do not know a priori the relationship between \(L^v_{post_i}\) and \(l(e, post_i)\) the disjunction is helpful.
Objective Function

Since we are interested in the most precise range information, we add an objective function to our constraint system. It minimizes the total number of values to be considered:

$$\min : \sum_{v \in V} \sum_{B_i \in R_v} (U^v_{pre_i} - L^v_{pre_i}),$$

where \( R_v = \{ B_i \in B \mid v \text{ is read in block } B_i \} \).
Solving the System of Symbolic Bound Constraints

Our system is almost linear except:

- We have both conjunctions and disjunctions.
- We compare first degree multivariable polynomials not just linear combinations of unknown rational variables.
Enclosing Disjunction via Integer Variables

We describe our approach using a small example. Consider the following constraint:

\[ L_{\text{pre}_j}^v \leq L_{\text{post}_i}^v \lor L_{\text{pre}_j}^v \leq l(e, \text{post}_i). \]

We introduce two new binary variables \( D_1 \) and \( D_2 \), and let \( M \) denote a large positive number. Our original constraint is then replaced with the following:

- \( L_{\text{pre}_j}^v - (M \cdot D_1 \sum_{p \in P} p_0) \leq L_{\text{post}_i}^v \)
- \( L_{\text{pre}_j}^v - (M \cdot D_2 \sum_{p \in P} p_0) \leq l(e, \text{post}_i) \)
- \( D_1 + D_2 \leq 1 \)

The new constraint system is stronger than the original one, and the two constraints are actually equivalent if \( M \) is sufficiently large. Note that the new variables \( D_1 \) and \( D_2 \) are the only variables that need to be pure integer variables, while all others can have rational values.
Handling Polynomials

Assume all variables in \( \{p_0 \mid p \in P\} \) are positive; then to satisfy

\[
X + \sum_{p \in P} X_p * p_0 \quad op \quad Y + \sum_{p \in P} Y_p * p_0
\]

it is enough to ensure that

\[
(X \quad op \quad Y) \quad \land \quad \bigwedge_{p \in P} (X_p \quad op \quad Y_p),
\]

where \( op \in \{=, <, >, \leq, \geq\} \).

Note that if both \( X \) and \( Y \) are linear expressions with unknown rational variables, then so is \( X \quad op \quad Y \).
Symbolic Bounds Analysis Example (1/2)

```c
int find(elType * arr, int n, elType el) {
    int i = 0;
    while (i < n) {
        if (arr[i] == el)
            return i;
        i = i + 1;
    }
    return i;
}
```

Objective Function:

\[
\min : (U_8^i - L_8^i)
\]

Initialization Constraints:

\[
L_1^i = -\infty \quad U_1^i = +\infty \quad L_1^n = n_0 \quad U_1^n = n_0
\]

Assignment Constraints:

\[
L_2^i = 0 \quad U_2^i = 0 \quad L_7^i = L_6^i + 1 \quad U_7^i = U_6^i + 1
\]

Flow Constraints:

\[
\begin{align*}
L_3^i & \leq L_7^i & U_3^i & \geq U_7^i \\
L_8^i & \leq L_3^n & U_8^i & \geq U_3^i \\
L_j^i & \leq L_{j-1}^i & U_j^i & \geq U_{j-1}^i \\
L_k^n & \leq L_{k-1}^n & U_k^n & \geq U_{k-1}^n
\end{align*}
\]

\( j \in \{3, 5, 4(L), 6, 8\} \quad k \in \{2, 3, 4(U), 5, 6, 7, 8\} \)

Handling Conditionals:

\[
\begin{align*}
(U_4^i & \geq U_3^n - 1) \lor (U_4^i & \geq U_3^n) \\
(L_4^n & \leq L_3^i + 1) \lor (L_4^n & \leq L_3^i) \\
(L_8^i & \leq L_3^n) \lor (L_8^i & \leq L_3^n) \\
(U_8^n & \geq U_3^i) \lor (U_8^n & \geq U_3^i)
\end{align*}
\]

(d) Constraints
Symbolic Bounds Analysis Example (2/2)

Consider the following constraint:

\[(L^i_8 \leq L^n_3) \lor (L^i_8 \leq L^i_3)\]

Assuming \(n_0 \geq 0\) and the following bound expressions for \(L^i_8, L^n_3\), and \(L^i_3\):

\[L^i_8 = C_1n_0 + C_2, \quad L^n_3 = C_3n_0 + C_4, \quad L^i_3 = C_5n_0 + C_6,\]

we add the following linear constraints:

- \(C_1 - 2^{33} \cdot D_1 \leq C_3, \quad C_2 - 2^{33} \cdot D_1 \leq C_4\)
- \(C_1 - 2^{33} \cdot D_2 \leq C_5, \quad C_2 - 2^{33} \cdot D_2 \leq C_6\)
- \(D_1 + D_2 \leq 1\)

Solving MLP problem gives the following bounds for \(i\) at location 8:

\[L^i_8 = n_0, \quad U^i_8 = n_0\]
Main observation:

The range information, if used only in a BMC run of depth $k$, does not have to be sound for all computations of a program, but only for the traces up to length $k$.

As an example, consider the following code:

```c
int i = 0, j = readInput(); while (i < j * j) { i ++; }
```

We can set the upper bound of $i$ to $k$, while no sound range analysis can do that.
Computing Bounded Ranges

- **The algorithm.**

  Initialize all bounds (coefficients) to the least conservative values;
  \[\text{for}(i = 0; i < k; i++)\]
  \[\text{foreach basic block } B_j\]
  \[\text{foreach variable } v, v \in V_f\]
  \[\text{update } L_{v \text{ pre}_j}, U_{v \text{ pre}_j}, L_{v \text{ post}_j}, \text{ and } U_{v \text{ post}_j} \text{ using constraints}\]

- **Free disjunctions and support for non-linear functions.** First, notice that resolving disjunctions "on-the-fly" is trivial. In addition, in case a function does not have any parameters, we can easily extend the algorithm to support many important non-linear functions. Consider an assignment \( y = x^2 \) in a block \( B_i \). The following rules can be used to update \( L_{y \text{ post}_i} \) and \( U_{y \text{ post}_i} \): \( L_{y \text{ post}_i} = 0 \) and \( U_{y \text{ post}_i} = \max(|L_{x \text{ pre}_i}|, |U_{x \text{ pre}_i}|)^2 \).

- **Increasing precision.** This algorithm always gives better bounds than LP-based approach. However, the precision can be further increased, by sacrificing the running time.
## Experiments

### Model Reduction:

<table>
<thead>
<tr>
<th>Bench mark</th>
<th>No RA bits</th>
<th>No RA gates</th>
<th>MLP bits</th>
<th>MLP gates</th>
<th>BoundedRA bits</th>
<th>BoundedRA gates</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPP</td>
<td>1435</td>
<td>24628</td>
<td>445</td>
<td>18955</td>
<td>258</td>
<td>17909</td>
</tr>
<tr>
<td>TCAS</td>
<td>1481</td>
<td>8792</td>
<td>765</td>
<td>5550</td>
<td>336</td>
<td>3912</td>
</tr>
<tr>
<td>BKRY</td>
<td>198</td>
<td>1515</td>
<td>24</td>
<td>449</td>
<td>24</td>
<td>449</td>
</tr>
<tr>
<td>ARRY</td>
<td>359</td>
<td>5212</td>
<td>160</td>
<td>2440</td>
<td>42</td>
<td>1369</td>
</tr>
<tr>
<td>Total</td>
<td>3473</td>
<td>40147</td>
<td>1394</td>
<td>27394</td>
<td>660</td>
<td>23639</td>
</tr>
</tbody>
</table>

### Running Time Reduction:

<table>
<thead>
<tr>
<th>Bench mark</th>
<th>No RA SAT</th>
<th>No RA BDD</th>
<th>MLP SAT</th>
<th>MLP BDD</th>
<th>BoundedRA SAT</th>
<th>BoundedRA BDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPP</td>
<td>TO(69)</td>
<td>TO(169)</td>
<td>TO(89)</td>
<td>TO(216)</td>
<td>TO(93)</td>
<td>TO(273)</td>
</tr>
<tr>
<td>TCAS</td>
<td>2.1</td>
<td>278.7</td>
<td>1.7</td>
<td>109.6</td>
<td>1.7</td>
<td>40.1</td>
</tr>
<tr>
<td>BKRY</td>
<td>70.2</td>
<td>16.1</td>
<td>27.7</td>
<td>0.6</td>
<td>27.7</td>
<td>0.6</td>
</tr>
<tr>
<td>ARRY</td>
<td>3.1</td>
<td>47.1</td>
<td>1.1</td>
<td>7.8</td>
<td>0.2</td>
<td>3.3</td>
</tr>
<tr>
<td>Total(bottom 3)</td>
<td>75.4</td>
<td>341.9</td>
<td>30.5</td>
<td>118</td>
<td>29.6</td>
<td>44</td>
</tr>
</tbody>
</table>
Conclusion

Summary of the talk:

- We have presented an improved version of the range analysis framework proposed by Radu Rugina and Martin Rinard. One of our main contributions is increased precision due to the introduction of disjunctions.

- We have described the concept of bounded range analysis and demonstrated the usefulness of both methods for bounded model checking.

To conclude:

- Static analysis is useful if not essential for successful software model checking tools.

- Both LP and MLP solvers are useful and efficient in analyzing software-based problems and can “outperform” BDD/SAT solvers.