Symbolic Execution with Interval Constraint Solving and Meta-Heuristic Search

Mateus Borges, Marcelo d'Amorim (Fed. U. Pernambuco, Brazil)
Saswat Anand (Georgia Inst. of Technology)
David Bushnell, Corina Pasareanu (TRACLabs and CMU/NASA Ames)

April, 2012
Software errors are expensive

Software is everywhere

Annual cost of software errors to the US economy is \(~60B\) dollars [NIST2002]
Main approaches for finding errors

• Software Testing
  – Run code and observe effects

• Static Analysis
  – Analyze code without running it
Research Goal

Develop new techniques and improve existing techniques for Software Testing and Static Analysis
Symbolic Execution (SE)

• Analysis of programs with unspecified inputs
  – Execute a program on symbolic inputs
  – Introduced in the 70s by L. Clarke, J. King

• Symbolic states represent sets of concrete states

• For each path, build a path condition
  – Condition on inputs for execution to follow that path
  – Check satisfiability of path condition
    • Explore only feasible paths

• Symbolic state
  – Symbolic values/expressions for variables, path condition, and program counter
Code that swaps 2 integers

```c
int x, y;
if (x > y) {
    x = x + y;
y = x - y;
x = x - y;
if (x > y)
    assert false;
}
```

Concrete Execution Path

1. **x = 1, y = 0**
2. **1 > 0 ? true**
3. **x = 1 + 0 = 1**
4. **y = 1 - 0 = 1**
5. **x = 1 - 1 = 0**
6. **0 > 1 ? false**
Example: Symbolic Execution

Code that swaps 2 integers:

```c
int x, y;
if (x > y) {
    x = x + y;
    y = x - y;
    x = x - y;
    if (x > y)  
        assert false;
}
```

Symbolic Execution Tree:

```
path condition

[PC:true] x = X, y = Y

[PC:true] X > Y ?

false
[PC:X≤Y]END

true
[PC:X>Y] x = X + Y

[PC:X>Y] y = X + Y - Y = X

[PC:X>Y] x = X + Y - X = Y

[PC:X>Y] Y > X ?

false
[PC:X>Y ∧ Y ≤ X] END

true
[PC:X>Y ∧ Y > X] END

False!

Solve path conditions → test inputs
```
State-of-the-Art

- Explosion in number of tools and techniques
  - **Tools**: CUTE, CREST, DART, EGT, EXE, jCUTE, KLEE, PEX, PREFIX, SAGE, SMART, SPF, YOGI, etc.
  - **Industry**: Bell Labs, Fujitsu, IBM, Microsoft, etc.
  - **Government Labs**: NASA, etc.
  - **Universities**: Berkeley, EPFL, Illinois, Imperial College London, Iowa State, U Nebraska Lincoln, UT Austin, UT Arlington, Stanford, Stellenbosch, etc.
  - **Research communities**: software engineering, programming languages, systems, security, etc.
Challenge

Handling complex mathematical constraints

Example constraint generated with SE for a module from TSAFE (Tactical Separation Assisted Flight Environment)

\[
\sqrt{\text{pow}\left(\left((x_1 + (e_1 \times (\cos(x_4) - \cos(x_4 + (((1.0 \times (((c_1 \times x_5) \times (e_2/c_2))/x_6)) \times x_2/e_1)))) - (((e_2/c_2)) \times (1.0 - \cos((c_1 \times x_5))))), 2.0\right)\right)} > 999.0 \quad \& \quad (c_1 \times x_5) > 0.0 \quad \& \quad x_3 > 0.0 \quad \& \quad x_6 > 0.0 \quad \& \quad c_1 = 0.017... \quad \& \\
\begin{align*}
c_2 &= 68443.0 \quad \& \quad e_1 = \left(\left(\text{pow}(x_2, 2.0) / \tan((c_1 \times x_3))\right) / c_2\right) \quad \& \\
e_2 &= \text{pow}(x_6, 2.0) / \tan(c_1 \times x_3)
\end{align*}
\]
CORAL: CONSTRAINT SOLVER FOR COMPLEX CONSTRAINTS
CORAL

• Target application of solver: SE of programs that
  – Use floating-point arithmetic
  – Call specific math functions

Input: sqrt(pow(((x1 + (e1 * (cos(x4) – ...

Output: \{x1=100.0, x2=98.48..., x3=3.08...E-11, ...

Approach: combine meta-heuristic search and interval solving
Meta-Heuristic Search

• Explores candidate solutions
  – Start with random solutions
  – Refine candidate set based on *fitness function*

Inherently incomplete!
Meta-Heuristic Search

• Local search: Uses **one** single candidate solution
  – E.g., Alternating Variable Method (AVM), hill climbing, simulated annealing, etc.

• Global search: Uses **several** candidate solutions
  – E.g., Particle Swarm Optimization (PSO), genetic algorithms, etc.
Interval Solving

• Another method of constraint solving
• Applications in several domains. E.g., stability in control systems.

**Input:** \( \text{sqrt} \left( \text{pow}(\ldots) \right) \)

**Output:** \{ \( x_1 = [99.9..., 100.0] \), \( x_2 = [99.9..., 100.0] \), \ldots \}, \ldots

Intervals may not contain solutions!
Our Approach: Combine Techniques

Meta-heuristic search

+ Good for finding exact solutions in large search spaces
- May get lost in local maxima

Interval solving

+ Good for computing parts of solution space
- Does not compute solutions

Seed meta-heuristic search with inputs drawn from intervals
(intuition: better initial states)
Implementation

- For meta-heuristic search
  - Uses Opt4J (http://opt4j.sourceforge.net/)
- For interval solving
  - Uses RealPaver (RP) [Granvilliers & Benhamou, 2006]
- Also includes several optimizations
  - E.g., inference of variable domains, elimination of dependent variables, etc.
Tool

- Integrated with NASA’s Symbolic PathFinder (SPF)
  
  http://babelfish.arc.nasa.gov/trac/jpf/wiki/projects/jpf-symbc

- Also available as a library

  http://pan.cin.ufpe.br/coral

---

What is CORAL?

CORAL is a meta-heuristic constraint solvers for dealing with numerical constraints in mathematical functions.

**Target**

The goal of CORAL is to improve symbolic execution of numeric applications. Symbolic generate test input data. It requires a constraint solver component to solve the constraint program. Certain classes of constraints admit a (decision) procedure that can determ
EVALUATION
Subjects

- Publicly available applications from the aerospace domain

<table>
<thead>
<tr>
<th>Subject</th>
<th># constraints</th>
<th># conjuncts</th>
<th># functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apollo Autopilot</td>
<td>800</td>
<td>39</td>
<td>3</td>
</tr>
<tr>
<td>Collision Detection (CDx)</td>
<td>800</td>
<td>63</td>
<td>6</td>
</tr>
<tr>
<td>Conflict Probe</td>
<td>33</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Turn Logic</td>
<td>329</td>
<td>3</td>
<td>20</td>
</tr>
</tbody>
</table>

TSAFE units
Evaluated CORAL Configurations

• Meta-heuristic search alone
  – AVM
  – PSO (previously found it better than GA)

• Interval solving w/ RealPaver (RP) alone
  – RP+RAN (choose random values from interval)

• Combinations of IS with global and local search
  – RP+AVM – optimistic on RP reported intervals
  – RP+PSO – not so optimistic
Results for Apollo and CDx

**APOLLO**
- PSO: 95
- RP+AVM: 0
- RP+PSO: 261
- RP+RAN: 120

**CDx**
- PSO: 73
- RP+AVM: 372
- RP+PSO: 231
- RP+RAN: 104
Distance from Solution to RP Box

The diagram shows the distribution of distance from the solution to the RP box for different methods: "RP + PSO", "RP + AVM", "mean RP + PSO", and "mean RP + AVM". The x-axis represents the number of datapoints (successful solve requests), while the y-axis represents the distance from the solution to the reference box.
Many solutions within intervals
Distance from Solution to RP Box

RP + AVM solutions closer to intervals than RP + PSO
Time Results

• Solvers typically respond very fast
• The most extreme case: ~40s
• We limit the # of iterations not execution time
• Random solving slower than others (10^6 iterations vs 600 PSO)
Conclusions for CORAL

• Combination solved more constraints than meta-heuristic search or interval solving alone
  – Both global and local search help interval solving
  – Complementary: should be run together in parallel

http://pan.cin.ufpe.br/coral
Related Work

• Combining interval solving and SAT solving.
  – iSAT [Franzle et al., JSAT’10]

• Using meta-heuristic search.
  – FLoPSy [Lakhotia et al., ICTSS’10]

• Optimizing solving with constraint propagation.
  – Choco [Jussien et al., EMNTR’10]

• Use concrete values and randomization to simplify complex constraints.
  – DART [Godefroid et al., PLDI’05],
  – EXE [Cadar et al., CCS’06]
Conclusions

• CORAL
  – Solver for complex mathematical constraints generated by symbolic execution
  – Combines meta-heuristic search with interval solving
    • Techniques are effective and complementary

• Future Work
  – Robustness analysis [Majumdar & Saha, RTSS 2009]
  – Overflow and round-off error bounds analysis
THANK YOU!