Symbolic Robustness Analysis

Rupak Mujumdar & Indranil Saha
[RTSS 2009]

Presented by:
Sayali Salvi, Saarland University
Seminar on “Robustness of Hardware and Software Systems”
Outline

- Control Systems
- Robustness
- Symbolic Robustness Analysis
- Algorithm and Implementation
- Limitations
Control Systems

Environment

Uncertain and imprecise

Safety-Critical Embedded Systems

Functional Safety

Safety-critical systems: a malfunctioning of the system can cause significant damage and may lead to loss of life. Non-compliance is indication for negligence in liability suits.
Robustness

We say system is robust, when small perturbations in the system inputs cause only small changes in its outputs.

Controller design

Controller implementation

Robust

Question: Is the implementation still Robust?
Robustness

System is $(\delta, \varepsilon)$-robust in input $x_1$
Transmission Calculation Example

int calc_trans_slow_torques (int angle, int speed)
{
    int pressure1, pressure2;
    int gear, val1, val3, val4;
    val1 = lookup1 (&(data_table[0][0]), angle);
    if (3 * speed <= val1)
        gear = 3;
    else
        gear = 4;
    val3 = lookup2 (&(out1_table[0][0]), gear);
    pressure1 = val3 * 1000;
    val4 = lookup2 (&(out2_table[0][0]), gear);
    pressure2 = val4 * 1000;
}

angle = 30, speed = 13
pressure2 = 0

angle = 30, speed = 14
pressure2 = 1000
Robustness Analysis

• Why is it difficult?
  • huge input space to be tested exhaustively
  • many different code execution paths
  • many control computations based on table lookups

• Why is it required?
  • Random testing ineffective

• This paper studies robustness analysis for control software: using ‘Symbolic Execution’
Symbolic Execution

- Test generation technique

- Executes program on *symbolic inputs*
  e.g. angle = ang, speed = sp

- Collects symbolic constraints along each execution path
  e.g. consider 2 paths computing pressure2 in our example
Symbolic Execution on our example

```c
int calc_trans_slow_torques (int angle, int speed)
{
    int pressure1, pressure2;
    int gear, val1, val3, val4;
    val1 = lookup1 (&(data_table[0][0]), angle);
    if (3 * speed <= val1)
        gear = 3;
    else
        gear = 4;
    val3 = lookup2 (&(out1_table[0][0]), gear);
    pressure1 = val3 * 1000;
    val4 = lookup2 (&(out2_table[0][0]), gear);
    pressure2 = val4 * 1000;
}
```

**data_table:**

<table>
<thead>
<tr>
<th>angle</th>
<th>val1</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>41</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**out2_table:**

<table>
<thead>
<tr>
<th>gear</th>
<th>val4</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Path1 symbolic constraints:

\[ \text{ang} \leq 30 \land \text{val1} = 41 \land 3 \times \text{sp} \leq \text{val1} \land \text{gear} = 3 \land \text{val4} = 0 \land \text{pressure2} = \text{val4} \times 1000 \]
Symbolic Execution on our example

int calc_trans_slow_torques (int angle, int speed)
{
    int pressure1, pressure2;
    int gear, val1, val3, val4;
    val1 = lookup1 (&(data_table[0][0]), angle);
    if (3 * speed <= val1)
        gear = 3;
    else
        gear = 4;
    val3 = lookup2 (&(out1_table[0][0]), gear);
    pressure1 = val3 * 1000;
    val4 = lookup2 (&(out2_table[0][0]), gear);
    pressure2 = val4 * 1000;
}

angle = 30, speed = 14

Path2 symbolic constraints:
ang \leq 30 \land val1 = 41 \land 3 \ast
sp > val1 \land gear = 4 \land val4
= 1 \land pressure2 = val4 \ast
1000
Concolic Execution

- Symbolic testing technique

Concolic = Concrete + Symbolic

- Symbolic constraints for each explored path at the end of concolic execution

- Concolic Execution
  - Concrete i/p and Symbolic i/p

- Symbolic output and path constraints

- Constraint Solver
  - New Concrete i/p and Symbolic i/p
Concolic Execution on our example

```c
int calc_trans_slow_torques (int angle, int speed)
{
  int pressure1, pressure2;
  int gear, val1, val3, val4;
  val1 = lookup1 (&(data_table[0][0]), angle);
  if (3 * speed <= val1)
    gear = 3;
  else
    gear = 4;
  val3 = lookup2 (&(out1_table[0][0]), gear);
  pressure1 = val3 * 1000;
  val4 = lookup2 (&(out2_table[0][0]), gear);
  pressure2 = val4 * 1000;
}
```

Concrete input: angle = 30, speed = 13
Symbolic input: angle = ang, speed = sp
Symbolic path constraint: 3 * sp <= val1

Negate the path conditional constraint:
Symbolic path constraint: 3 * sp > val1
Solve the modified constraints:
New concrete input: angle = 30, speed = 14
Symbolic input: angle = ang, speed = sp
Symbolic path constraint: 3 * sp > val1

Concrete input: angle = 30, speed = 14
Symbolic input: angle = ang, speed = sp
Symbolic path constraint: 3 * sp > val1
Symbolic Robustness Analysis

1. Path1 symbolic constraints:
   \[
   \text{ang} \leq 30 \land \text{val1} = 41 \land 3 \times \text{sp} \leq \text{val1} \land \\
   \text{gear} = 3 \land \text{val4} = 0 \land \text{pressure2} = \text{val4} \times 1000
   \]

2. Path2 symbolic constraints:
   \[
   \text{ang}' \leq 30 \land \text{val1}' = 41 \land 3 \times \text{sp}' > \text{val1}' \\
   \land \text{gear}' = 4 \land \text{val4}' = 1 \land \text{pressure2}' = \text{val4}' \times 1000
   \]

- Formulate the optimization problem for above two paths:
  Maximize \(|\text{pressure2} - \text{pressure2}'|\)
  subject to the constraints:
  
  1. Path1 symbolic constraints
  2. Path2 symbolic constraints
     \[
     \text{angle} = \text{angle}' \\
     |\text{speed} - \text{speed}'| \leq 1
     \]

- Iterate over all path pairs to find the maximum deviation in output (pressure2) for a perturbation of 1 unit (δ) in the input (speed).
Formal Problem Definition

\[
\varepsilon_{yx} = \max_{v, x, x'} \left\{ \left| y - y' \right| \mid \begin{array}{l}
y = P(v, x) \\
y' = P(v, x') \\
| x - x' | \leq \delta_x \\
\end{array} \right\}
\]

\(x\) : actual input value
\(x'\) : measured input value
\(v\) : value of all other input variables
\(y\) : value of output of program \(P\) for input \(v, x\)
\(y'\) : value of output of program \(P\) for input \(v, x'\)
\(\delta_x\) : at most deviation of input variable \(x\)
\(\varepsilon_{yx}\) : maximum deviation in the value of output variable \(y\)

Program \(P\) is \((\delta, \varepsilon)\)-robust in input \(x\) if \(\varepsilon_{yx} \leq \varepsilon\) when \(\delta_x \leq \delta\)
Implementation Algorithm

Algorithm find_output_sensitivity \((P, x, \delta_x)\)

\[
S = \text{concolic}(P);
\]

\[
\epsilon_{yx} = 0;
\]

for \((e_1, \xi_1)\) in \(S\) do

for \((e_2, \xi_2)\) in \(S\) do

\[
\Delta = \text{find_output_deviation}(e_1, \xi_1, e_2, \xi_2, \delta_x);
\]

\[
\epsilon_{yx} = \max(\epsilon_{yx}, \Delta);
\]

end

end

return \(\epsilon_{yx}\);

• Produces a set of pairs \((e, \xi)\) for each explored path
  - symbolic output expressions
  - path constraints

• Returns the solution of the optimization problem.
• Returns -1 when constraints cannot be satisfied for a pair of paths.
Implementation

• Concolic execution: **Splat**
  • Uses the decision procedure STP to solve the path constraints to generate inputs for a new execution
  • Is extended to generate the symbolic output expression and the path constraints for all program paths

• Optimization problem solving: **Lindo**
  • Provides general nonlinear and nonlinear/integer optimization problem solving APIs
  • Can detect if a set of constraints is not satisfiable
Limitations

- Limitations in algorithm and implementation:
  - Decision procedure handles fixed point numbers, not floating point numbers
  - $\delta_x$ is treated as constant
  - Closed loop systems can not be analyzed

In $n$ iterations: input $x_1$ will be $2^n\delta$ apart
Take Away

- Robustness analysis of control software program
- Symbolic technique of test case generation
Seminar

- Robustness analysis of various types of entities:
  - Control Software programs
  - General Software programs
  - Sequential circuits
  - Networked systems
  - Cache replacement policies
Idea

- Application of a approach used in robustness analysis of sequential circuits to analyze closed loop control software:
  - Sensor input variable values: actual and measured
  - Corresponding value sequences of state input variable
  - Corresponding value sequences of output variable
  - Common Suffix Distance:
    last position of difference = loop iteration number after which small change sensor input is forgotten