Symbolic Robustness Analysis

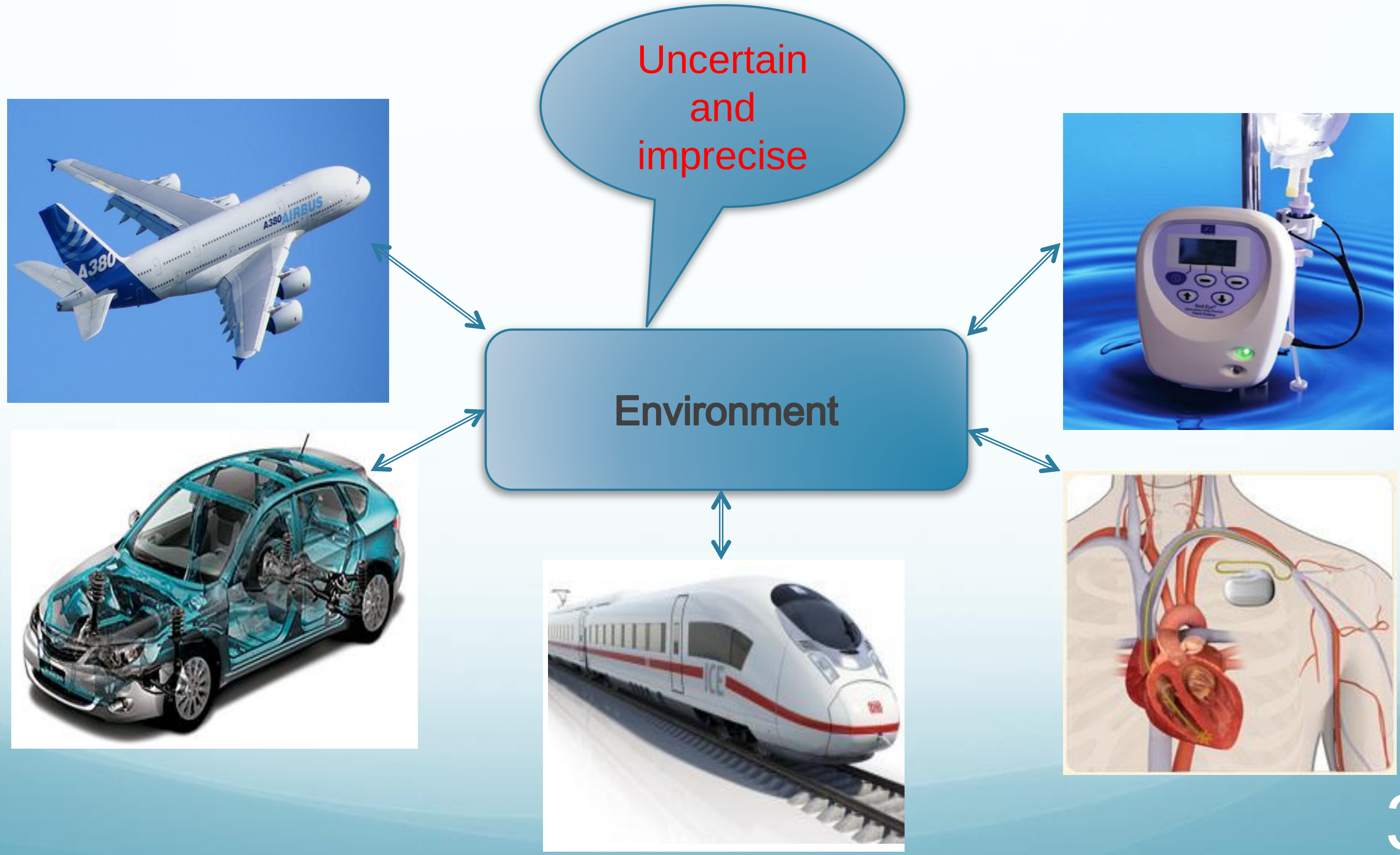
Rupak Mujumdar & Indranil Saha
[RTSS 2009]

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



Robustness

Controller design

Controller implementation

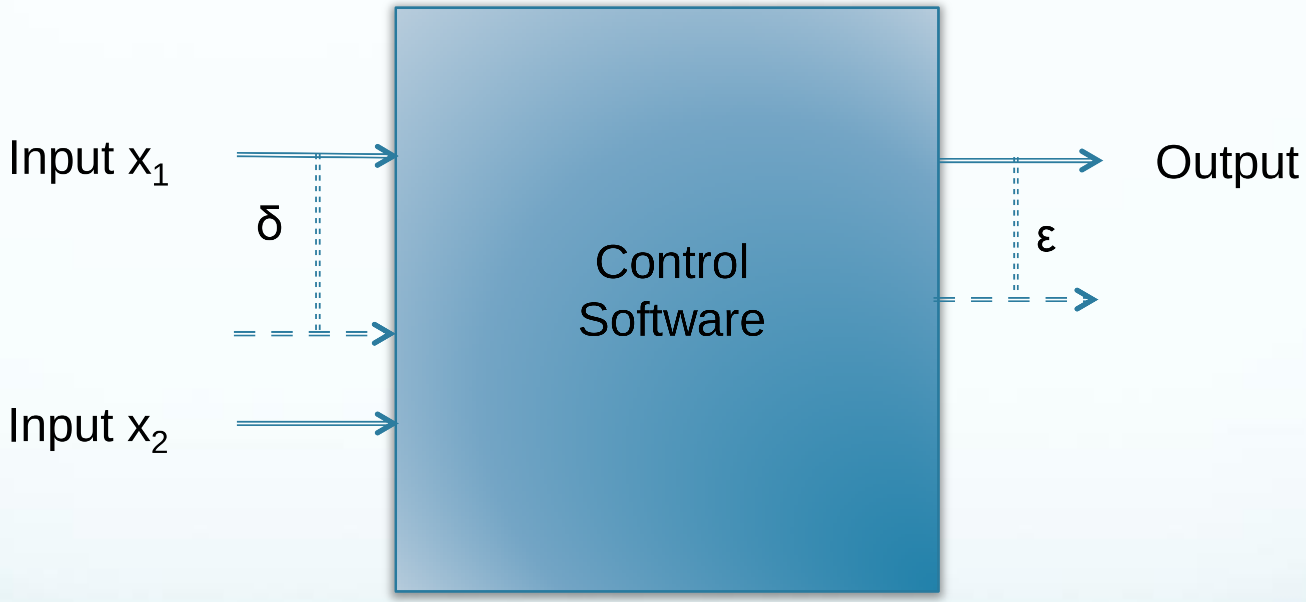
Robust

Question:
Is the implementation still Robust?

- We say system is robust, when

small perturbations in the system inputs cause only small changes in its outputs.

Robustness



System is (δ,ε)-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;
}
```

val1=41

gear=4

val4=1

data_table:

angle	val1
30	41
40	63
…	…

out1_table:

gear	val3
1	0
2	0
3	1
4	1

out2_table:

gear	val4
1	0
2	0
3	0
4	1

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

```
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

data_table:

angle	val1
30	41
…	…

out2_table:

gear	val4
…	…
3	0
4	1

Path1 symbolic constraints:

ang <= 30 ∧ val1 = 41 ∧ 3 * sp <= val1 ∧ gear = 3 ∧ val4 = 0 ∧ pressure2 = val4 * 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

data_table:

angle	val1
30	41
…	…

out2_table:

gear	val4
…	…
3	0
4	1

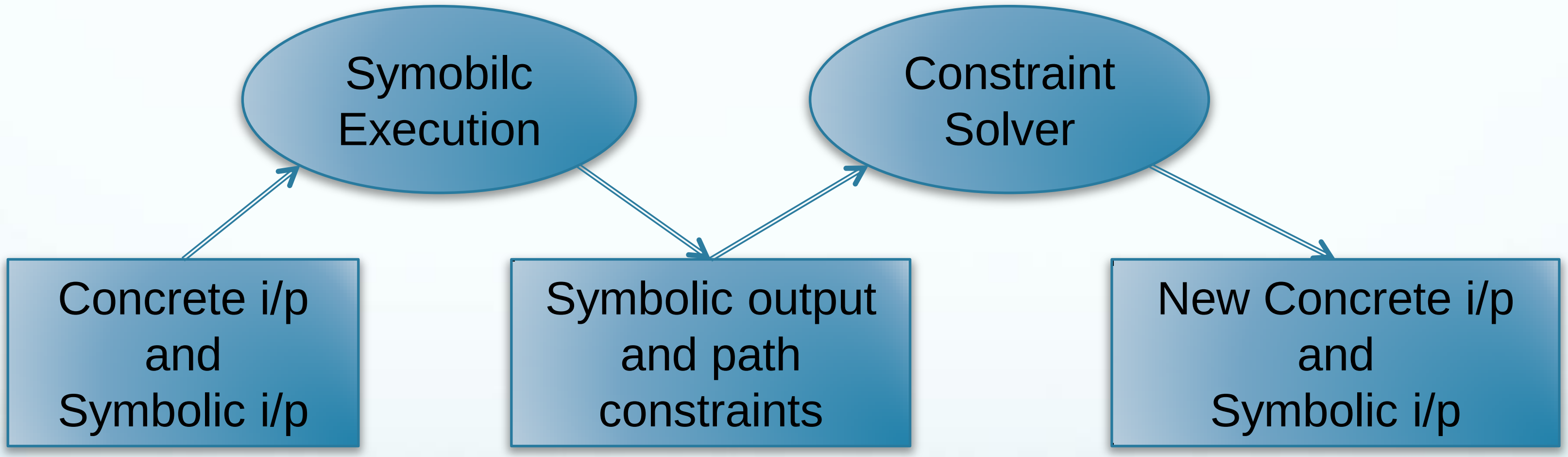
Path2 symbolic constraints:

ang <= 30 ∧ val1 = 41 ∧ 3 * sp > val1 ∧ gear = 4 ∧ val4 = 1 ∧ pressure2 = val4 * 1000

Concolic Execution

- Symbolic testing technique

  Concolic = Concrete + Symbolic



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

Concolic 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;
}
```

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

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:

ang <= 30 ∧ val1 = 41 ∧ 3 * sp <= val1 ∧ gear = 3 ∧ val4 = 0 ∧ pressure2 = val4 * 1000

2. Path2 symbolic constraints:

ang' <= 30 ∧ val1' = 41 ∧ 3 * sp' > val1' ∧ gear' = 4 ∧ val4' = 1 ∧ pressure2' = val4' * 1000

- Formulate the optimization problem for above two paths:

  Maximize |pressure2 − pressure2'|

  subject to the constraints:

  1. Path1 symbolic constraints
  2. Path2 symbolic constraints

  angle = angle'

  |speed − speed'| <= 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

$$\varepsilon_{yx} = \max_{v,x,x'} \left\{ |y - y'| \;\middle|\; \begin{array}{l} y = P(v,x) \\ y' = P(v,x') \\ |x - x'| \le \delta_x \end{array} \right\}$$

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'

δ_x : at most deviation of input variable x

ε_{yx} :maximum deviation in the value of output variable y

Program P is (δ,ε)-robust in input x if $\varepsilon_{yx} <= \varepsilon$ when $\delta_x <= \delta$

Implementation Algorithm

Algorithm find_output_sensitivity (P, x,δ_x)

S = concolic (P);

ε_{yx} = 0;

for (e_1,ξ_1) in S do

  for (e_2,ξ_2) …

    Δ = find … $_x$);

    ε_{yx} = m… (ε_{yx} …)

  end

end

return ε_{yx};

- **Produces a set of pairs (e,ξ) for each explored path**
  e symbolic output expressions
  ξ path constraints

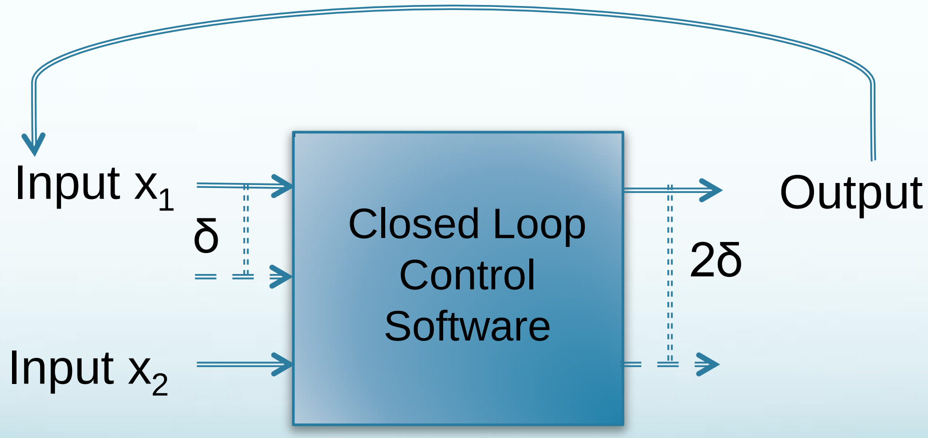
- **Returns the solution of the optimization problem.**
- **Returns -1 when constraints can not be satisfied for a pair of paths.**

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
  - δ_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

THANK YOU!