SNS COLLEGE OF ENGINEERING

Kurumbapalayam(Po), Coimbatore – 641 107
Accredited by NAAC-UGC with 'A' Grade
Approved by AICTE, Recognized by UGC & Affiliated to Anna University, Chennai

Department of Artificial Intelligence and Data Science

23ITT203 Object Oriented Software Engineering

Symbolic Execution

Introduction to Symbolic Execution

What is Symbolic Execution?

- Symbolic execution is a software testing technique where, instead of running a program with actual input values (like 5 or "abc"), we use symbolic variables (like x, y, or input1) to represent any possible input.
- This allows the program to explore all possible paths it could take depending on the input values.

Example:

- Traditional testing: test with x = 5Symbolic execution: test with x = any value, then check what the program would do in every case (x > 0, x = 0, x < 0)

Key Concepts

Term	Description
Symbolic Variable	A placeholder input (like x) representing any possible value
Path Condition	A set of conditions that must be true to follow a specific logic path in the program
Execution Path	A flow of program execution based on decisions (like if-else)
Constraint Solver	A tool that determines real inputs which satisfy the path condition
Path Explosion	Too many possible paths caused by many conditions or loops
Object State	The current data and behavior of an object in OOP (e.g., balance = 0)

How Symbolic Execution Works

- **Inputs as Symbols:** Inputs are not real numbers but variables like x, y, etc.
- **Branching:** Every decision in the code (if, switch) creates new paths.
- **Path Conditions:** For each path, collect the conditions needed to follow it.
- **Constraint Solving:** Use a solver to find real values that match the path conditions.
- **Test Generation:** Each valid path can be turned into an automatic test case.

Example

```
def check_number(x):
    if x > 5:
        return "Greater"
    else:
        return "Smaller or Equal“
```

Normal Execution Example:

Case 1: check_number(10) → "Greater“

Case 2: check_number(3) → "Smaller or Equal“

Symbolic Execution Example:

We say: Let x be any value, not a specific number.

We look at the program and follow both possible paths:

Path 1: Condition: x > 5

Output: "Greater“

Path 2: Condition: x <= 5

Output: "Smaller or Equal"

Comparison between Normal and Symbolic Exceution

Feature	Normal Execution	Symbolic Execution
Input Type	Real values (e.g., x = 10)	Symbols (e.g., x = anything)
Checks	One path at a time	All possible paths
Example Input	`x = 3`, `x = 10`, etc.	Just `x` (no value given)
Result	One output	Multiple outcomes with conditions
Useful For	Running code normally	Testing all possibilities & finding bugs

Why Symbolic Execution is Useful in OOP

In object-oriented systems:

- We deal with **objects**, each having different **states**
- Each object has **methods** that behave differently depending on inputs and state

Symbolic execution:

- Tests how methods behave with **all input combinations**
- Detects errors in logic early
- Validates how **object states** change over time

Advantages

- Finds bugs early in development
- Helps cover **all logic paths** (even rare ones)
- Generates test cases automatically
- Great for testing **object behaviors** and **interactions**

Limitations

- Can be slow or complex for large programs (due to path explosion)
- May not handle programs with dynamic or external inputs well
- Limited for systems that rely heavily on user interface or hardware


thank
you