SYMBOLIC EXECUTION

- A POWERFUL TECHNIQUE FOR SOFTWARE ANALYSIS AND VERIFICATION

LIMITATIONS OF TRADITIONAL TESTING

01 Traditional testing relies on concrete inputs.

02 Debugging can be challenging when a test fails, requiring tracing the execution path.

03 Manual test case generation is time-consuming and prone to overlooking critical scenarios.

04 Difficult to achieve high code coverage, especially for edge cases and error handling.

WHAT IS SYMBOLIC EXECUTION?

- Uses symbolic variables instead of concrete values.
- Symbolically executes the program, maintaining variable expressions.
- Explores multiple execution paths simultaneously.
- Generates path constraints, which define conditions for specific execution paths.

HOW SYMBOLIC EXECUTION WORKS

- Assign symbolic variables to inputs.
- Execute statements, updating symbolic expressions.
- Fork execution at conditional statements for each possible outcome.
- Add path constraints for each branch taken.
- Use a solver to check path constraint feasibility.
- Stop when all paths are explored or a limit is reached.

EXAMPLE - SYMBOLIC EXECUTION IN ACTION

```c
int foo(int a, int b) {
  if (a > 5) {
    if (b < 10) return a + b;
      else  return a - b;
  } else return a * b;
}
```

APPLICATION - AUTOMATED TEST GENERATION

01 **Achieves high code coverage with minimal manual effort.**

02 **Discovers edge cases and boundary conditions automatically.**

03 **Generates test suites targeting specific program paths or behaviors.**

APPLICATION - VULNERABILITY DETECTION

01 Identifies buffer overflows, integer overflows, and memory safety issues.

02 Detects flaws in input validation or sanitization.

03 Analyzes security-critical sections for potential exploits.

APPLICATION - FORMAL VERIFICATION

- Proves absence of errors like division by zero or null pointer dereference.
- Verifies adherence to specified program properties.
- Provides formal guarantees, critical for safety-sensitive systems.

KEY BENEFITS OF SYMBOLIC EXECUTION

- Systematic exploration of program paths.
- Achieves high code coverage.
- Detects bugs early in the development cycle.
- Reduces manual effort through automation.
- Pinpoints errors with precise path constraints.

CHALLENGES AND LIMITATIONS

- Path explosion due to exponential growth of paths.
- Complex constraints can be computationally expensive.
- Difficult to model external functions and libraries.
- Loops with symbolic bounds can lead to infinite paths.
- Challenges in modeling interactions with the environment.

THE FUTURE OF SYMBOLIC EXECUTION

- Improving scalability through concolic execution and state merging.
- Developing efficient and specialized SMT solvers.
- Integrating symbolic execution with fuzzing and machine learning.
- Expanding applications to smart contracts, cyber-physical systems, and AI safety.
- Enhancing user-friendly tools for broader adoption.

CONCLUSION AND Q&A

- SYMBOLIC EXECUTION IS A POWERFUL APPROACH TO SOFTWARE ANALYSIS AND VERIFICATION.
- IT ACHIEVES HIGHER COVERAGE AND EARLY BUG DETECTION THAN TRADITIONAL METHODS.

- ONGOING RESEARCH IS ADDRESSING CHALLENGES AND EXPANDING APPLICATIONS.
- ENCOURAGES EXPLORATION OF TOOLS AND STUDIES FOR DEEPER INSIGHTS.

TEAM MEMBERS

- سيد مصطفى عطية
- شريف طلعت فوزى
- احمد مصطفى محمد
- محمد امين حسن
- احمد حسن محمود