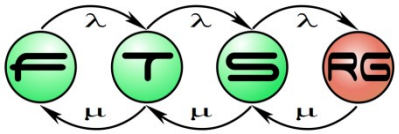
Testing solutions for LabVIEW applications

Áron Szabó

Advisor: Dr. Zoltán Micskei

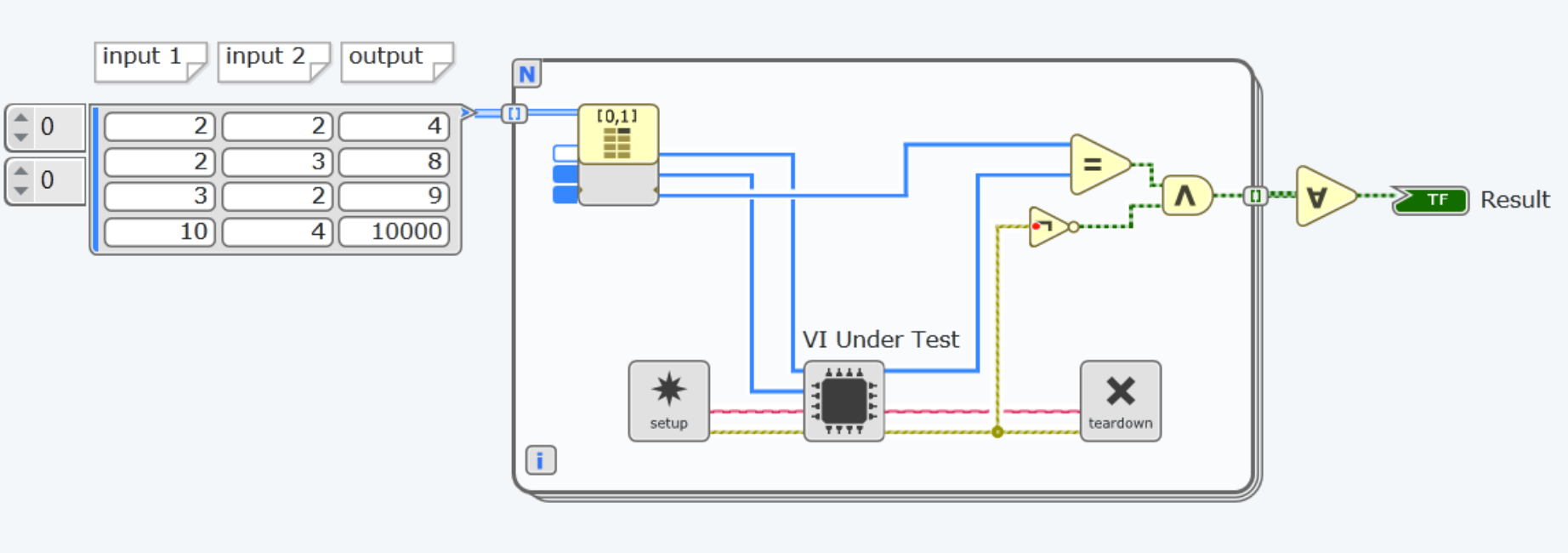
Budapest University of Technology and Economics
Fault Tolerant Systems Research Group



Unit testing and LabVIEW

- **Unit testing** is testing of individual hardware or software units or groups of related units
- **LabVIEW** is a system engineering software tool
  - rapid development and prototyping
  - testing systems, measurement solutions
  - other industrial applications.
- Unit testing for LabVIEW applications?

Unit testing in LabVIEW



- Existing LabVIEW testing frameworks
- Creating tests resource-effectively
- Test generating solutions

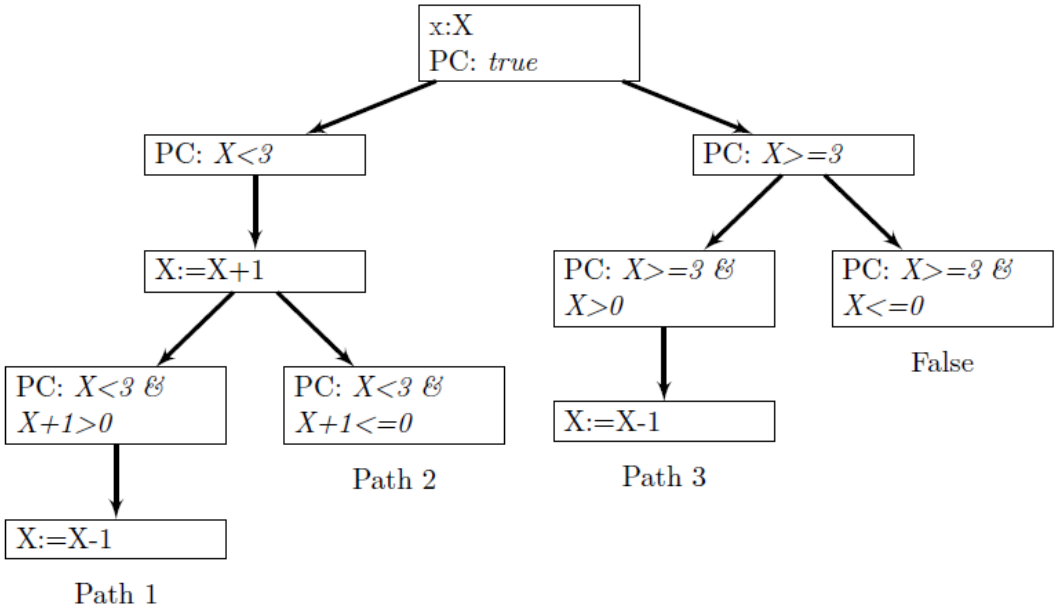
Designing the tool

- **Goal:** a unit test generation tool for LabVIEW VIs
- In the **scope** of BSc thesis: define test inputs using symbolic execution
- With a **limited** feature set: few essential operators and program structures
- **Output:** list of execution paths, their path conditions and calculated symbolic variables

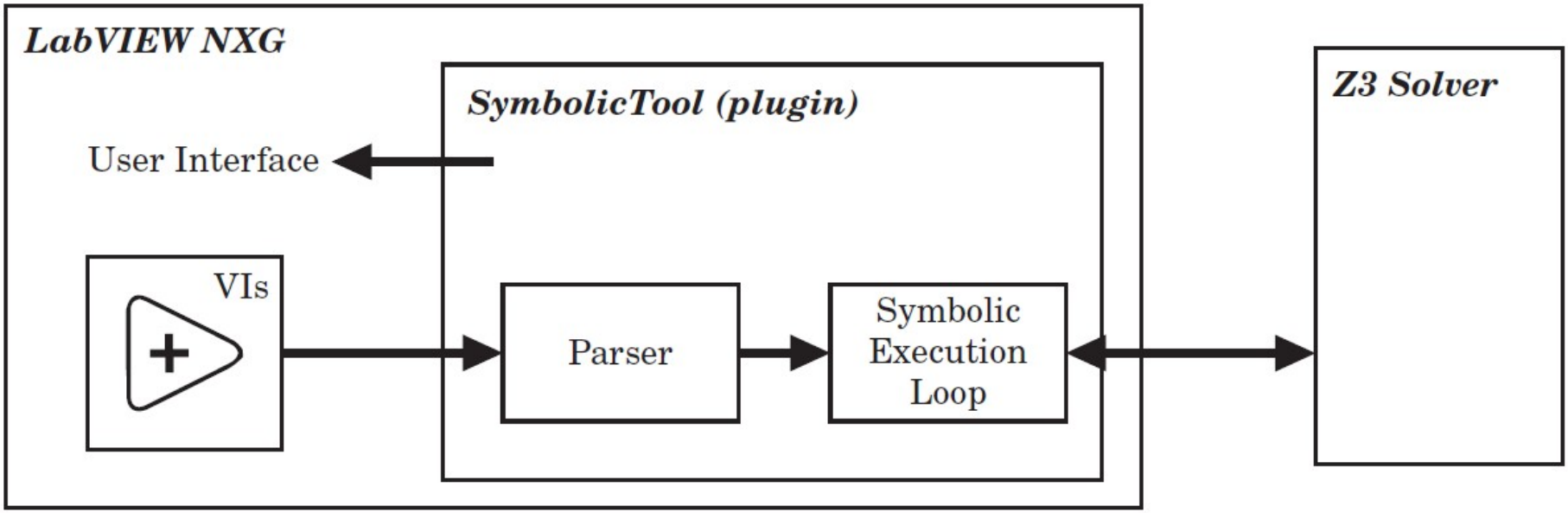
Symbolic Execution

- Running the program with symbolic input variables
- Discover all possible execution paths
- Define sets of input variables to reach an execution path

```
void test(int x){
   if(x<3)  x=x+1;
   if(x>0)  x=x-1
   return x;
}
```



Architecture



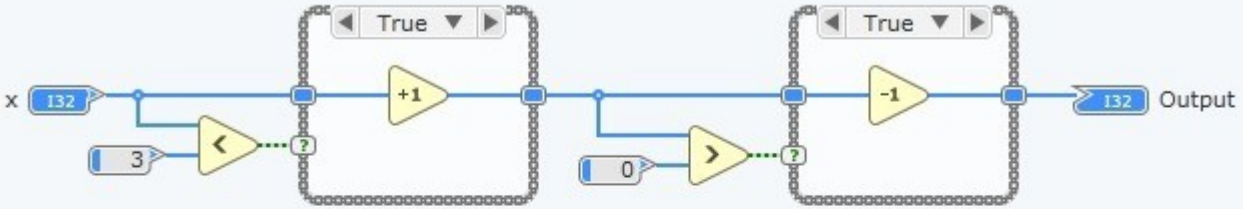
- Conversion of the LabVIEW program:
  data-flow VIs → imperative representation
- Symbolic Execution Loop
- Constraint solver to calculate the input values from each path constraint

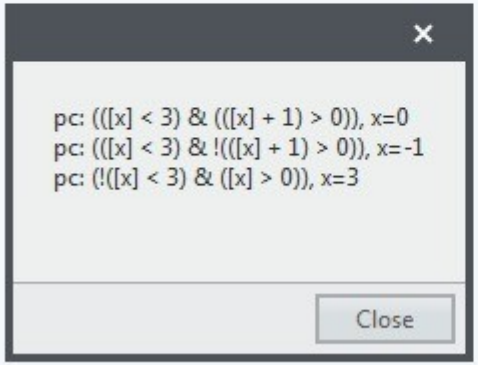
Implementation

- LabVIEW NXG: .NET API for object model, plugin
- Tool written in C#
- **Parser**
  - BFS-like algorithm on LabVIEW model
  - building an internal object model
- **Symbolic Execution Tree**
  - fork on branching statements
  - finished executions in leafs
  - DFS-like algorithm
- Constraint Solver: Call Microsoft Z3 Solver

Evaluation

```
A = [x];
B = 3;
C = 0;
D = (A < B);
IF D Then
E = (A + 1);
ELSE
E = A;
End IF
F = (E > C);
IF F Then
G = (E - 1);
ELSE
G = E;
End IF
[Output] : G;
```





- Successful execution for demo LabVIEW programs
- All defined requirements satisfied
- Supports basic operators and structures

Summary

- Unit-testing and LabVIEW
- Cost-effective testing of industrial applications
- Symbolic Execution
- Design and implementation of the tool
  - Parser: data-flow → imperative
  - Symbolic Execution Loop
  - Constraint Solver

Question 1

- **There is a sentence in your thesis: *„One of them was automatic unit test generation, which in my opinion is not very effective on its own: generating tests with some inputs and exercising the output (which comes from an execution of the program) says very little.”* How does your solution help overcome this concern?**
- Without an additional technology (mutation testing, symbolic execution), generating random tests is inefficient
- In most cases a human is still needed to watch over the operation of the tool (filtering, association to requirements)

Question 2

- **Is there any plan to make the product available to wider audience (end users or developers)?**
- A complete solution might be influential on LabVIEW unit testing
- Further development or release of source code

Question 3

- **Is there any particular reason you put all the operators in one class, instead of deriving a new class for each?**
- Result of quick prototyping (few differences between operators)
- Definitely needs refactoring

```
switch (Op)
{
    case "+":
        return new Constant((a as Constant).Value + (b as Constant).Value);
    case "-":
        return new Constant((a as Constant).Value - (b as Constant).Value);
    case "*":
        return new Constant((a as Constant).Value * (b as Constant).Value);
    case "<":
```