

Project Report — Formal Methods Tool

Course: Formal Methods (Spring 2025)

Instructor: Ms. Nigar Azhar Butt

Team Members:

- Abdul Rafay — 22I-8762
- Sumeed Jawad –22I-2651
- Ubaidullah – 22I-1248

Section: B

Objective

To build a GUI-based tool that helps verify correctness and check equivalence of programs using **Static Single Assignment (SSA)** transformation and **SMT solving with Z3**.

Features

- Parses simple imperative programs in a custom mini-language
 - Converts to SSA form with optimization
 - Generates SMT-LIB constraints
 - Verifies correctness (assert statements)
 - Checks semantic equivalence of two programs
 - Supports loop unrolling (user-defined depth)
 - Displays SSA, SMT, and results (with counterexamples) in a web-based GUI
-

Language Syntax

The tool supports a custom mini-language with:

- Assignments: $x := \text{expr}$
- If-else: `if (cond) { ... } else { ... }`
- While loops: `while (cond) { ... }`

- For loops: `for (init; cond; update) { ... }`
 - Array initialization: `arr := [3, 1, 2]`
 - Assertions: `assert(expr)`
 - Loop-range assertions:
`assert(for (i in range(0, n-1)): arr[i] <= arr[i+1]);`
-

SSA Translation

- All variables are versioned: `x1, x2, ...`
 - Control flow uses *phi functions* to merge values across branches
 - Loop unrolling is applied before SSA
 - Optimizations:
 - Constant propagation
 - Dead code elimination
 - Common subexpression elimination
-

🔧 GUI Workflow

Modes:

- **Verification** (1 program)
- **Equivalence** (2 programs)

Steps:

1. Input code (and second code if in equivalence mode)
2. Specify unrolling depth (if loops present)
3. Tool shows:
 - SSA form
 - SMT constraints
 - Result (Correct, Incorrect, Equivalent, Not Equivalent)

- Counterexamples (if any)

Limitations

- No heap or pointer support
 - Unrolling must be sufficient for loops to terminate
-

Formal Methods Analysis

Mode: Equivalence ▾

Code:

```
x := 0;
while (x < 4) {
  x := x + 1;
}
assert(x == 4);
```

Other Code (for equivalence):

```
x := 0;
while (x < 4) {
  x := x + 1;
}
assert(x == 4);
```

Unroll Depth: 2

Analyze

Result:

Programs are NOT equivalent (within unroll depth 2)

Formal Methods Analysis

Mode: Equivalence ▾

Code:

```
x := 0;
while (x < 4) {
  x := x + 1;
}
assert(x == 4);
```

Other Code (for equivalence):

```
x := 0;
while (x < 4) {
  x := x + 1;
}
assert(x == 4);
```

Unroll Depth: 4

Analyze

Result:

☒ Programs are equivalent (within unroll depth 4)
☒ Postcondition holds in both programs

Formal Methods Analysis

Mode: Verification ▾

Code:

```
x := 0;
while (x < 4) {
  x := x + 1;
}
assert(x == 4);
```

Unroll Depth: 7

Analyze

Result:

☒ Postcondition holds
☒ No counterexamples found within unroll depth 7

Formal Methods Analysis

Mode: Verification ▾

Code:

```
x := 3;
if (x < 5) {
  y := x + 1;
} else {
  y := x - 1;
}
assert(y > 0);
```

Unroll Depth: 2

Analyze

Result:

☒ Postcondition holds
☒ No counterexamples found within unroll depth 2