Formal Methods in Software Engineering

Project- **VeriTool**

**Submitted by:**

Huda Nyazee (22i-8790)

Umer Khalil (22i-2608)

Moawiz Bin Ammar (22i-2608)

**Formal Methods Verification Tool Report**

**1. Overview**

The **Formal Methods Verification Tool** is designed to analyze program correctness and equivalence using static analysis techniques. It enables users to input their programs, which are then parsed, transformed into Static Single Assignment (SSA) form, and verified against specified assertions.

**2. Language Syntax and Parser Assumptions**

**2.1 Input Program Syntax**

The tool accepts a simplified programming language syntax with the following constructs:

* **Variable Assignment**: x := 5;
* **Control Structures**: if, else, for, and while statements.
* **Assertions**: assert(condition); to specify postconditions.

**2.2 Postcondition Assertion Format**

* **Basic Conditions**:
* assert(condition);

where condition is a boolean expression.

* **Array Assertions**:  
  Direct assertions on arrays are not supported. Users can assert array conditions using indices:
* assert(array[i] == value);

**2.3 Parser Assumptions**

* Input programs must be syntactically correct.
* Nested control structures and basic arithmetic operations are supported.
* Comments are ignored, and whitespace is not significant.

**3. SSA Translation Logic**

The SSA (Static Single Assignment) translation includes:

* **Variable Renaming**: Each variable assignment generates a new version, e.g., x becomes x\_0, x\_1, etc.
* **Phi Functions**: Used at control flow join points to merge variable versions from different execution paths.
* **Expression Processing**: Recursively updates expressions to ensure correct version usage.

**4. Unrolling Handling**

* **Loop Unrolling Depth**: Users specify how many iterations of a loop should be unrolled during analysis.
* **Control Flow Representation**: The tool builds a **Control Flow Graph (CFG)** to represent unrolled loops for precise loop behavior analysis.

**5. SMT Formulation Strategy**

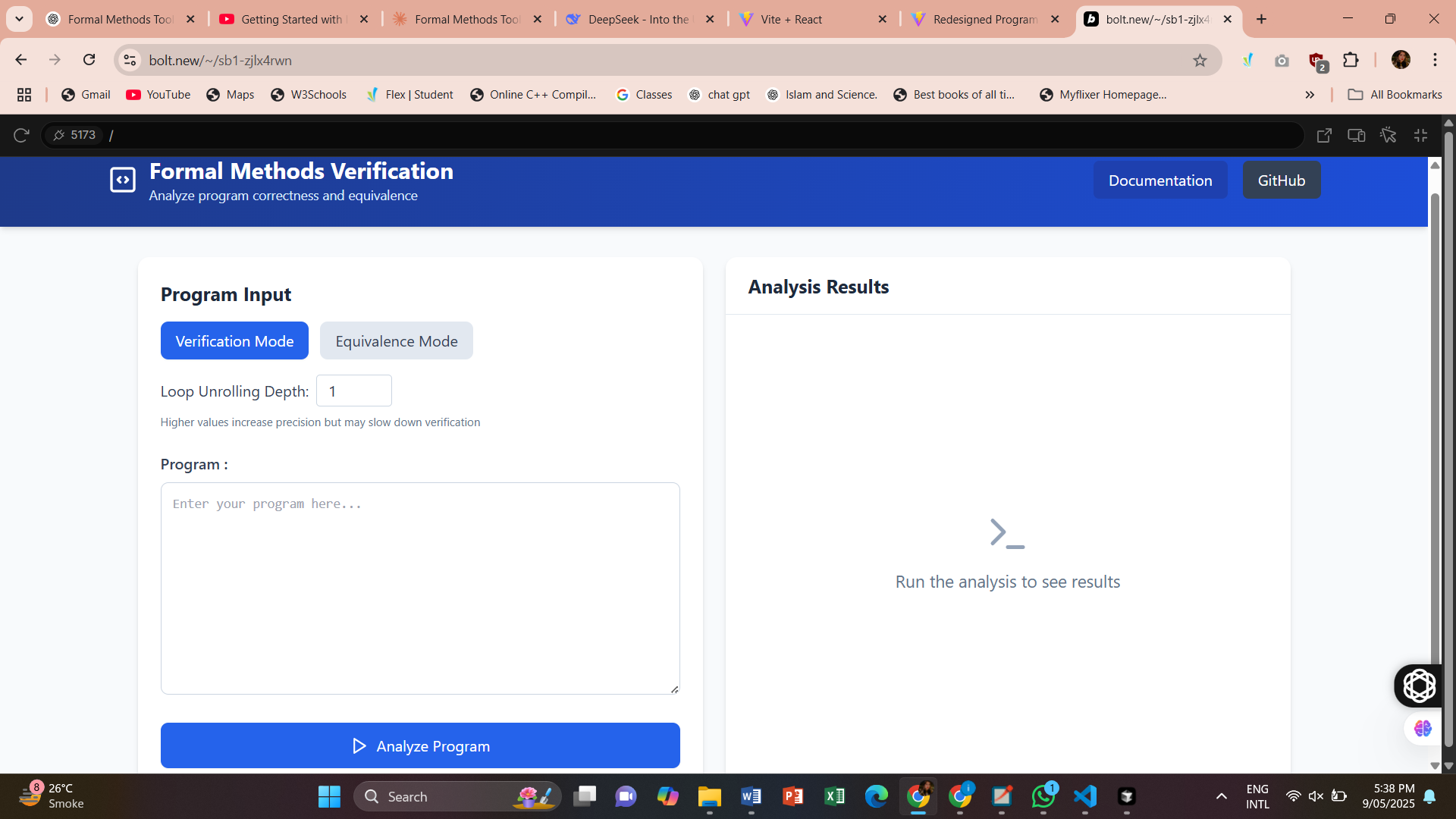
The SSA-translated program is converted into SMT (Satisfiability Modulo Theories) constraints:

* **Variable Declarations**: All variables are treated as integers.
* **Assertions**: Program assertions are translated to SMT assertions.
* **Control Flow Logic**: Represented using conditional expressions and phi functions.

**6. GUI and Test Results**

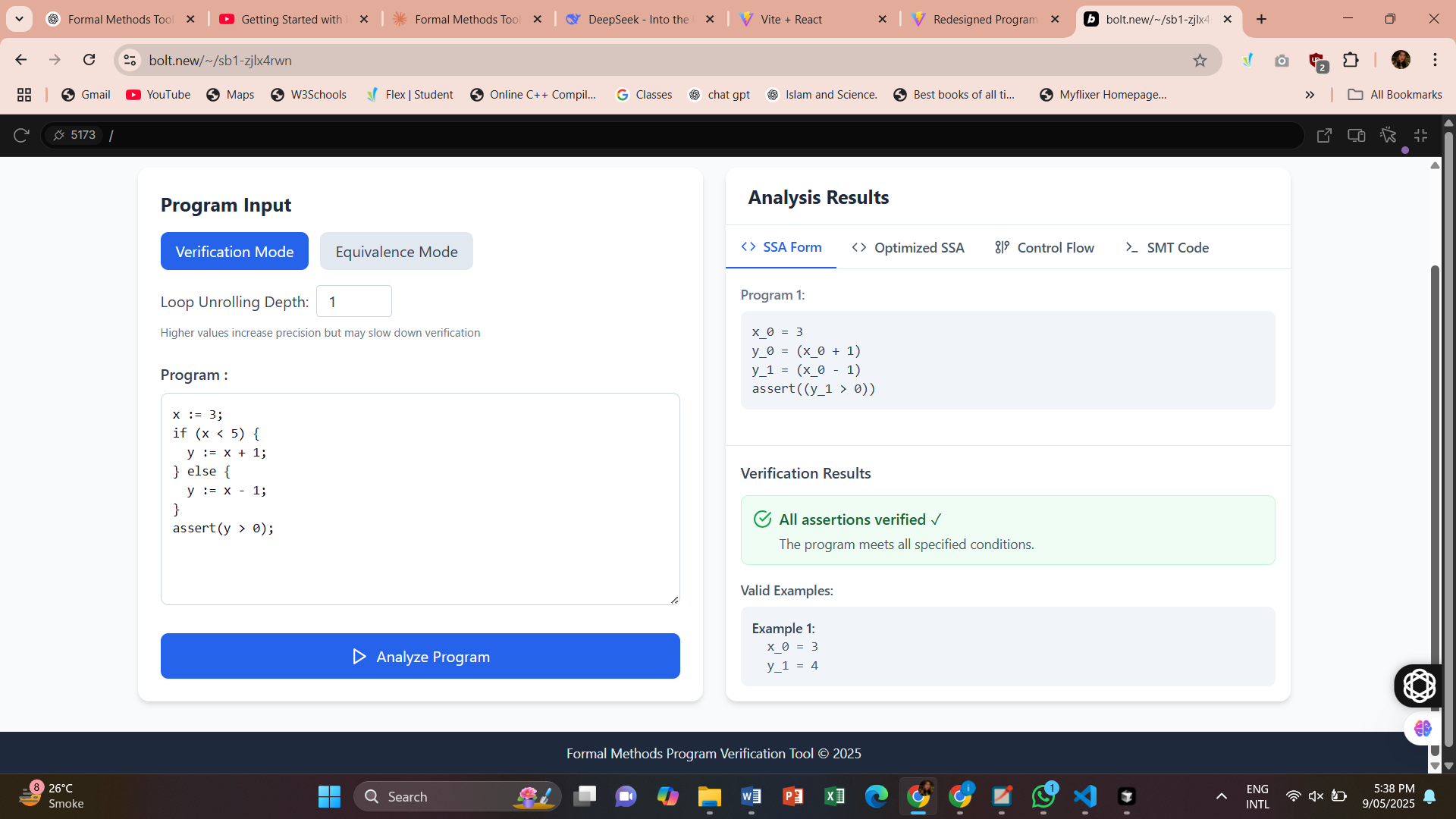
**6.1 Screenshots**

* The main interface for program input.

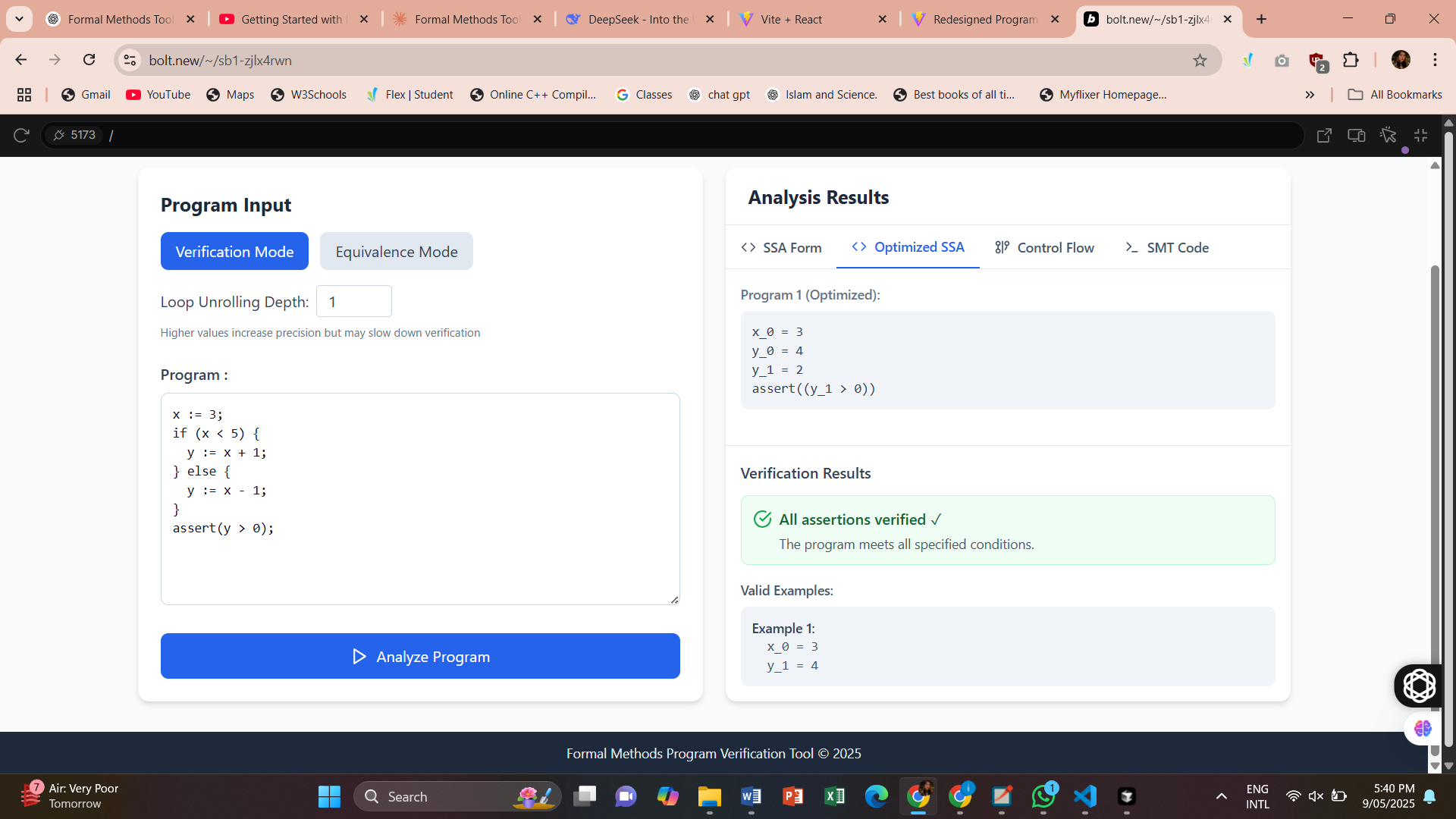


* SSA form, SMT code, and verification result outputs.

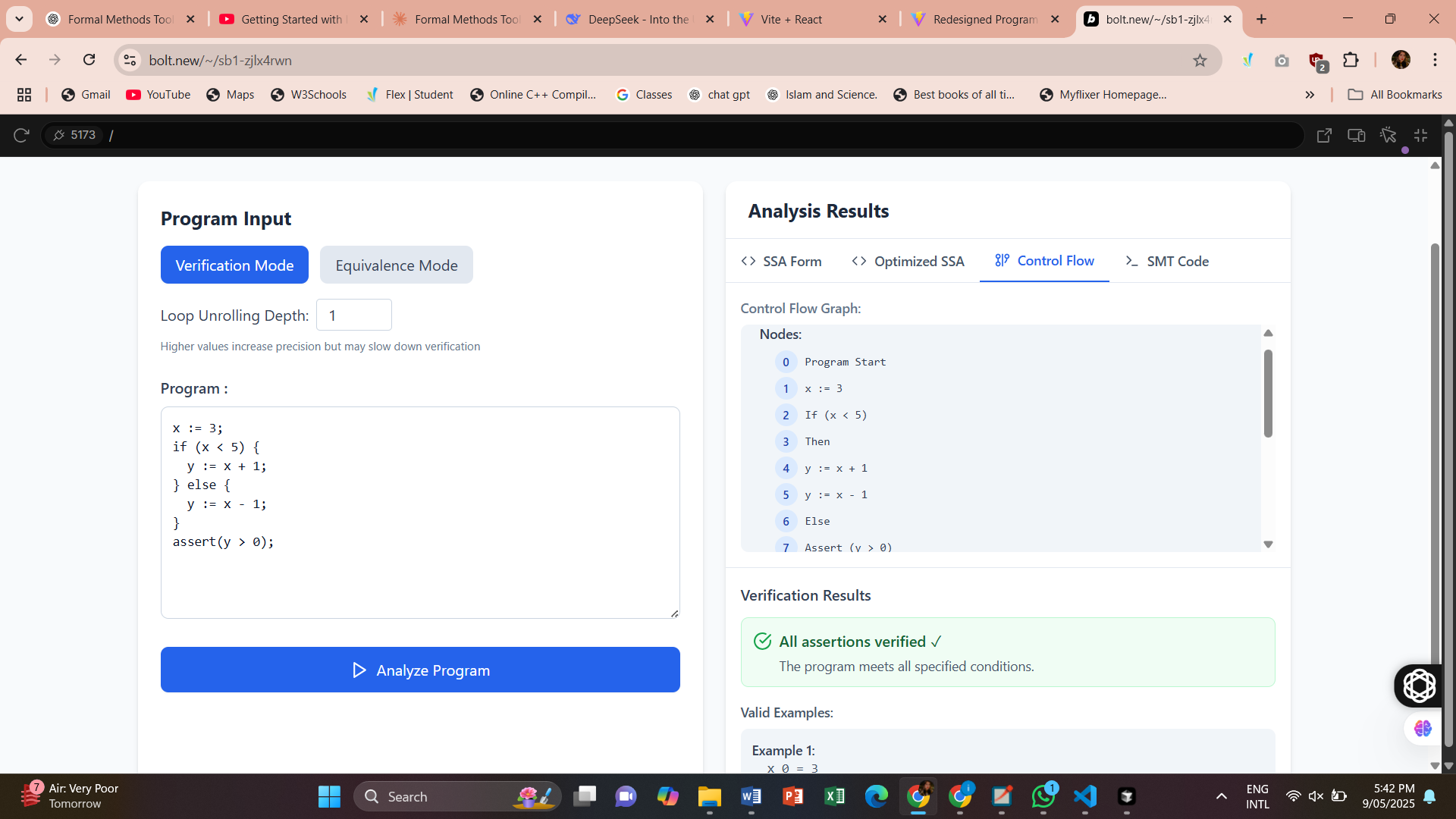
**SSA Form**

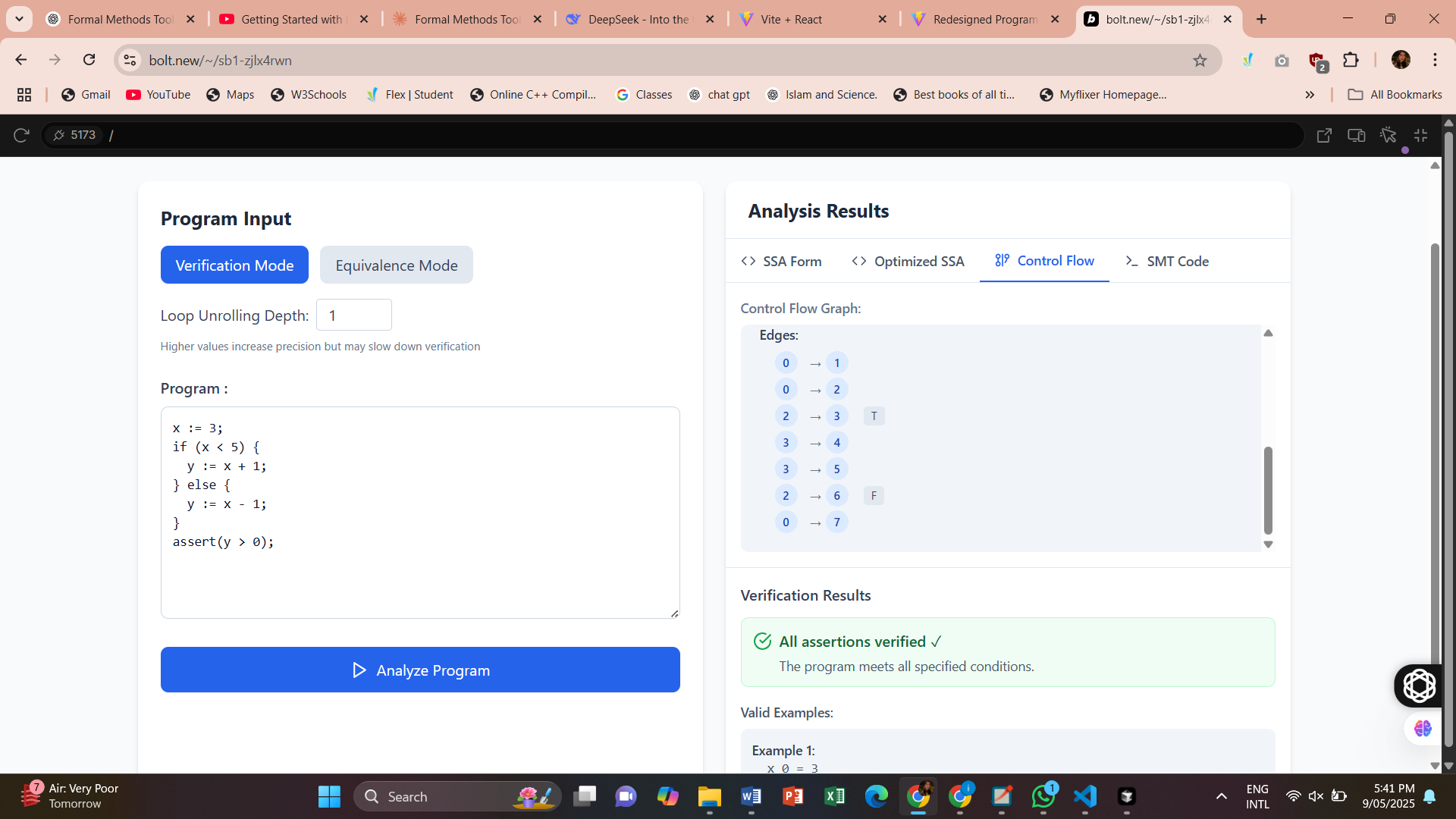


**Optimised SSA**



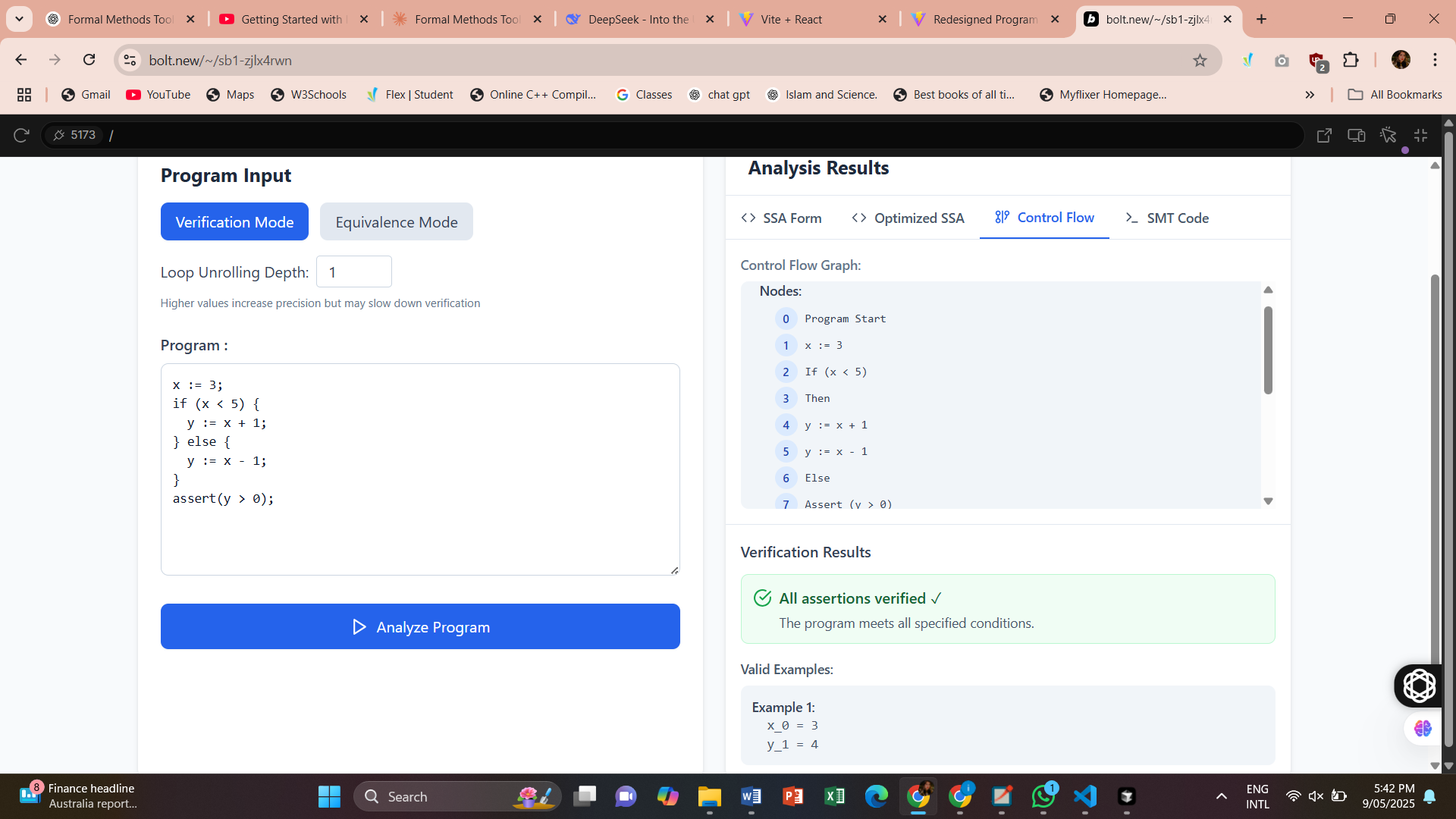
* Control flow graphs for visualizing program flow.



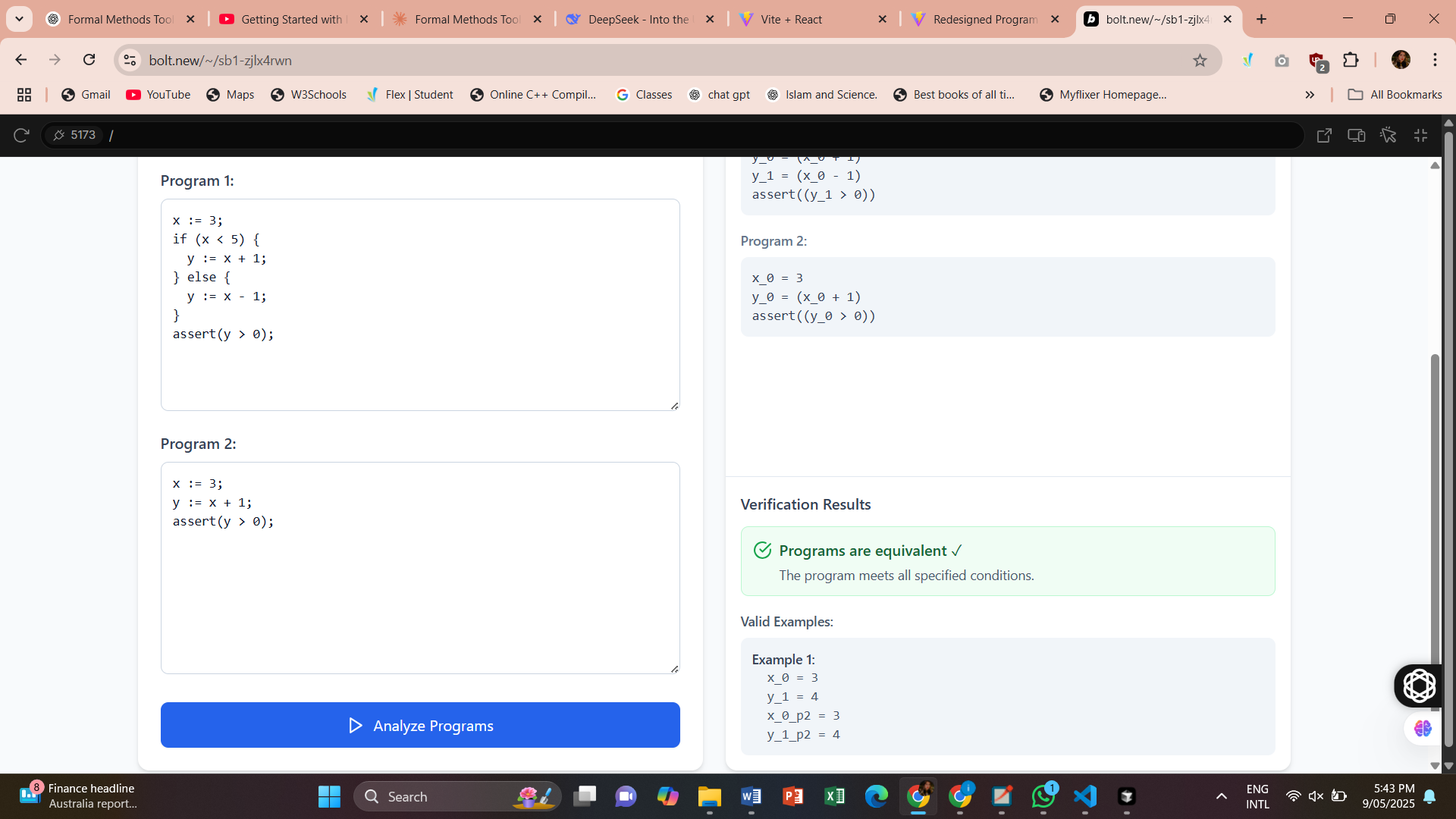


**6.2 Test Results**

* **Verification Tests**: Correctly verified all sample programs.



* **Equivalence Tests**: Correctly identified both equivalent and non-equivalent program pairs.



**7. Limitations and Improvements**

**7.1 Limitations**

* **Array Assertions**: Limited support for direct array assertions.
* **Error Handling**: Minimal feedback on syntax errors.
* **Performance**: May degrade with large programs or deep nesting.

**7.2 Improvements**

* **Enhanced Array Support**: Add native support for array-based assertions.
* **Improved Error Reporting**: Provide detailed syntax feedback.
* **Optimization Techniques**: Reduce SMT constraint size and improve performance.

**8. Component Breakdown**

**8.1 User Interface (UI)**

* **File**: project/src/components/VerificationPanel.tsx – Inputs and control actions.
* **File**: project/src/components/ResultsPanel.tsx – Displays SSA, SMT, and results.
* **File**: project/src/components/Header.tsx – Navigation and branding.

**8.2 Parser**

* **File**: project/src/utils/verificationUtils.ts – parseProgram constructs AST.

**8.3 SSA Converter**

* **File**: project/src/utils/verificationUtils.ts – convertToSSA performs transformation.

**8.4 CFG Generator**

* **File**: project/src/utils/verificationUtils.ts – generateCFG builds control flow.

**8.5 SMT Formulator**

* **File**: project/src/utils/verificationUtils.ts – generateVerificationSMT, generateEquivalenceSMT.

**8.6 SMT Solver Interface**

* **File**: project/src/utils/verificationUtils.ts – checkVerification, checkEquivalence.

**8.7 Results Processor**

* **File**: project/src/components/ResultsPanel.tsx – Interprets SMT solver output.

**8.8 Utilities**

* **File**: project/src/utils/verificationUtils.ts – Expression evaluation, formatting, and error utilities.

**8.9 Configuration and Settings**

* **File**: project/src/components/VerificationPanel.tsx – User-defined parameters (e.g., loop unrolling depth).

**9. Summary**

This tool enables formal verification through modular components:

* Parsing and SSA translation.
* CFG and SMT generation.
* SMT solver interaction.
* GUI-driven analysis output.

The structured architecture ensures flexibility for enhancements, such as improved data structure support, better error handling, and more efficient SMT encodings.