# S-Emulator: User Manual for Testing

**Student Information:**  
**Name:** Avshalom Sharabani  
**ID:** 318799608  
**Email:** Avshalomsh@mta.ac.il  
**GitHub:** <https://github.com/AvshiSha/S-emulator>

**Bonus:** Saving and loading the system

## Table of Contents

1. [System Overview](#overview)
2. [Key Design Decisions](#decisions)
3. [Bonus Features](#bonus)
4. [Usage Guide](#usage)
5. [Technical Implementation](#technical)

## 1. System Overview

### Core Functionality

* **Program Loading:** Load XML-based S-Instruction programs
* **Program Expansion:** Expand synthetic instructions to basic instructions
* **Program Execution:** Execute programs with input values and cycle counting
* **Visual Display:** Pretty-print programs and expansion results
* **State Management:** Save and load exercise states (Bonus Feature)

## 2. Key Design Decisions

### 2.1 Modular Architecture (Engine vs UI)

**Decision:** Separated core engine logic from the user interface.

**Rationale:**

* **Separation of Concerns:** Engine handles computation, UI handles interaction
* **Maintainability:** Changes to UI don't affect core logic
* **Testability:** The Engine can be tested independently

### 2.2 Java Serialization for Save/Load

**Decision:** Used Java's built-in serialization for state persistence

**Rationale:**

* **Simplicity:** No external dependencies required
* **Object Graph Preservation:** Maintains all object relationships
* **Type Safety:** Automatic type checking during deserialization
* **Performance:** Efficient binary format

### 2.3 Degree mechanism

**Decision:** Each expansion degree gets fresh row numbering (1, 2, 3, ...), and the expansion is seen based on the degree the user is selecting. Showing the instruction that expands to the user's degree.  
The expansion that I found for every static instruction is:  
All the basic instructions are zero degrees.  
zero variable: expand to 1.  
Goto Label: expand to 1.  
Assignment: expand to 2.  
Constant Assignment: expand to 2.  
Jump Zero: expand to 2.  
Jump Equal Constant: expand to 3.  
Jump Equal Variable: expand to 3.

**Rationale:**

* **Clarity:** Each degree is self-contained and easy to understand
* **Consistency:** Row numbers always start from 1 in each degree (1-based)
* **User Experience:** More intuitive for users to follow

### 2.4 Creation Chains Display

**Decision:** Show complete instruction lineage with ">>>" separator

**Rationale:**

* **Traceability:** Users can see how each instruction was created
* **Value:** Demonstrates the expansion process, printing that Aviad asks from us.

## 3. Bonus Features

### 3.1 Save/Load State Functionality

**Feature:** Complete session persistence with automatic file management

#### Save State Capabilities:

* **Complete Session Persistence:** Saves the current program, all run history, and execution results
* **Automatic File Naming:** State files are automatically named based on the loaded program (e.g., "programName.state")
* **Directory Creation:** Automatically creates parent directories if they don't exist
* **Error Handling:** Robust error handling for permission issues and invalid paths
* **Quote Handling:** Automatically removes quotes from user input paths

#### Load State Capabilities:

* **Full Session Restoration:** Restores the exact program and complete run history
* **Seamless Continuation:** Resume work exactly where you left off
* **Cross-Session Persistence:** Maintain progress across multiple sessions
* **State Validation:** Automatic validation of loaded state files
* **Type Safety:** Proper casting and validation of loaded objects

#### Implementation Details:

* **Run History Tracking:** Preserves all execution history, including inputs, outputs, and cycle counts
* **User-Friendly Interface:** Simple menu options (6 for Save, 7 for Load) with clear prompts

## 4. Usage Guide

### 4.1 Main Menu

When you start the application, you'll see the following menu:

=== S-Emulator === 1) Load program 2) Show program 3) Expand 4) Run 5) History 6) Save program 7) Load program 8) Exit Choose [1-8]:

### 4.2 Menu Options Explained

|  |  |  |
| --- | --- | --- |
| **Option** | **Description** | **Usage** |
| 1) Load program | Load an XML program file | Enter the full path to an XML file (e.g., C:\path\to\program.xml) |
| 2) Show program | Display the loaded program | Shows program details, inputs, labels, and instructions |
| 3) Expand | Expand the program to the chosen degree | Choose expansion degree (0 to max degree available) |
| 4) Run | Execute program with input values | Enter input values separated by commas |
| 5) History | View execution history | Shows table of all previous runs |
| 6) Save program | Save current exercise state | Enter directory path (file named automatically) |
| 7) Load program | Load previously saved state | Enter full path to .state file |
| 8) Exit | Exit the application | Closes the program |

### 4.3 Step-by-Step Workflow

#### Basic Workflow:

1. **Load Program:** Choose option 1 and provide the XML file path
2. **View Program:** Use option 2 to see the original program
3. **Expand:** Use option 3 to see the expansion to degree 1 or 2, or 3
4. **Execute:** Use option 4 to run the program with inputs
5. **Save State:** Use option 6 to save your progress (Bonus)

### 4.4 Error Handling Tests

#### Test 1: Invalid File Path

**Objective:** Verify error handling for invalid paths

**Steps:**

1. Choose option 1 (Load program)
2. Enter an invalid path (e.g., "invalid.xml")
3. Verify the appropriate error message

**Expected Result:** Clear error message, application continues

#### Test 2: Invalid Input Values

**Objective:** Verify input validation works

**Steps:**

1. Load a program
2. Choose option 4 (Run)
3. Enter invalid input (e.g., "abc, def")
4. Verify the error message and retry prompt

**Expected Result:** Input validation with clear error messages

## 5. Technical Implementation

### 5.1 Key Algorithms

#### Expansion Algorithm

The expansion process follows these steps:

1. **Initialize:** Start with degree 0 (original program)
2. **Iterate:** For each expansion step (1 to target degree)
3. **Expand:** Replace synthetic instructions with basic instructions
4. **Track:** Maintain parent-child relationships for lineage
5. **Number:** Assign fresh row numbers for each degree

#### Execution Algorithm

Program execution follows these steps:

1. **Initialize:** Set up execution context with input values
2. **Build Label Map:** Create a mapping of labels to instruction indices
3. **Execute:** Process instructions sequentially
4. **Jump:** Handle conditional and unconditional jumps
5. **Track:** Count cycles and update variable states

### 5.2 Error Handling Strategy

#### Multi-Layer Error Handling

* **Input Validation:** Validate user input before processing
* **File Validation:** Check file existence and format
* **XML Validation:** Validate XML structure and content
* **Execution Safety:** Handle runtime errors gracefully
* **User Feedback:** Provide clear, actionable error messages

## Conclusion

This S-Emulator system demonstrates advanced concepts in program transformation and execution. The modular design, comprehensive error handling, and bonus features make it a robust and user-friendly tool for understanding instruction expansion and program execution.

**Key Strengths:**

* Clean, maintainable architecture
* Comprehensive error handling
* User-friendly interface
* Educational value through visual feedback
* Bonus features enhance usability