C to Promela code Converter

May 14, 2025

Atul Kharat (2023CSB1105) ,
Hetvi Bagdai (2023CSB1123) ,
Lakshay Kumar (2023CSB1132) ,
Nachiket Patil (2023CSB1136).

Instructor:
Dr. Jagpreet Singh

Summary:
This project presents the design and development of a **C to Promela code translator**, aimed at facilitating the formal verification of C programs using the **SPIN model checker**. The translator provides an automated mechanism for converting key elements of C code—including data declarations, control flow constructs, and function-like procedures—into their Promela counterparts, preserving the original program's semantics for accurate model-based analysis.
The translation logic is implemented in `maincode.cpp`, which leverages **regular expressions** to parse C code structures such as `if-else`, `for` and `while` loops, `switch-case` statements, arrays, and basic data types. These elements are systematically mapped into Promela constructs like `if...fi`, `do...od`, channels, and typed variables (`byte`, `bool`, `mtype`). The parser also performs intelligent formatting, including proper indentation and block management, to enhance code readability in the generated output.
A graphical interface built with `Python (CustomTkinter)` enables users to input C code, trigger translation, and visualize the Promela output and compilation logs. The GUI also supports features such as syntax highlighting, file loading/saving, and error feedback for missing files or compilation issues.
The current version supports a wide range of foundational C constructs and assumes well-structured input. While advanced C features such as pointers, dynamic memory management, and template programming are not fully implemented, the tool provides a robust framework for extending support in future iterations.
By bridging the gap between low-level programming and high-level verification models, this translator enables software engineers to apply formal verification techniques more efficiently, making it a practical and educational tool for system modeling and correctness assurance.

1. Introduction:

In an era where software systems are increasingly concurrent and distributed, verifying their correctness has become a fundamental necessity. **Promela** (Process Meta Language) is a modeling language designed for describing system behavior, particularly useful in verifying the logic of concurrent systems through the **SPIN model checker**. Promela provides support for message passing, non-determinism, synchronization primitives, and process interactions, making it well-suited for simulating and analyzing multi-threaded programs.
Despite its strengths, writing Promela models by hand for verifying C programs can be both time-consuming and error-prone. To bridge this gap, this project proposes an automated **C to Promela Code Translator**, which parses well-structured C code and converts it into semantically equivalent Promela code. This enables developers, researchers, and students to analyze and verify C code behavior using SPIN with minimal manual translation effort.

The translator is implemented in C++ and utilizes **regular expressions** to parse and transform typical C constructs, including:

- Conditional statements (`if-else`),
- Loops, Nested loops, (`for`, `while`),
- Functions, Recursive Functions,
- Malloc, struct, pointers,
- Switch statements,
- Variable declarations, arrays, and basic expressions.

The tool produces equivalent Promela code using core constructs such as `proctype`, `init`, `do...od`, `if...fi`, and channels. An interactive Python GUI, developed using `CustomTkinter`, allows users to enter C code, run the translator, and view both the generated Promela code and compiler messages in a structured interface.
The educational material provided in the included reference project archive (`Programming Paradigm project.pdf`) served as a foundational guide to Promela's syntax and semantics. It covers essential Promela features such as data types, control structures, channel communication, and process definitions, which were directly incorporated into the translation rules.
This project not only facilitates automatic translation but also serves as a learning tool for understanding how high-level programming constructs are mapped into concurrent modeling languages. It is a stepping stone toward building robust model-driven software engineering tools that promote correctness and formal verification.

2. Algorithms and Techniques Used:

1. **Lexical Analysis (Tokenization)**
   The input C code is parsed to identify keywords, identifiers, operators, and other tokens. This helps in understanding the structure and components of the C code.

2. **Parsing and Syntax Mapping**
   Custom parsing techniques are employed to analyze the structure of the C program and convert it into an abstract representation. This structure is then used to generate semantically equivalent Promela constructs.

3. **Keyword Replacement Algorithm**
   The translation logic uses a rule-based approach to replace C keywords and structures (such as `if`, `for`, `while`, `functions`) with their corresponding Promela equivalents (`if...fi`, `do...od`, `proctype`).

4. **Syntax Tree Traversal**
   Statements in the C code are analyzed and mapped node-by-node, with conditional blocks and loops being handled using tree traversal and substitution techniques.

5. **GUI Highlighting and Interaction**
   The Python GUI uses keyword highlighting for C and Promela syntax using regular expressions and Tkinter tag manipulation to improve code readability.

6. **File Handling and I/O Synchronization**
   The system reads the C code from an input file, processes it using a compiled C++ executable, and displays the generated Promela code in a read-only output box. Console logs are captured for runtime and compilation errors.

7. **Subprocess Management in Python**
   The GUI invokes the C++ translation engine via a subprocess, capturing standard output and error streams to relay status to the user.

3. Working of the Project: C to Promela Code Translator:

This project provides an automated system to translate basic C code constructs into Promela (Process Meta Language), which is primarily used for model checking with the SPIN tool. The translation process leverages regular expressions for syntax matching and replacement, combined with a user-friendly graphical interface to facilitate the conversion process. The project comprises two main components: a backend implemented in C++ and a frontend developed using Python.

3.1. Backend Logic: Parsing and Translation (`maincode.cpp`)

The core translation logic is implemented in C++ using the `<regex>` standard library. The backend performs the following operations:

1. **Reading the Input File:** The C++ program opens and reads the contents of a file named `input.c`. This file contains the user's C code that is to be translated into Promela.

2. **Line-by-Line Processing:** Each line of code is processed individually. Regular expressions (regex) are used to identify specific C language constructs, such as:

    - **Variable declarations:** Matches patterns like `int a;` or `float b = 5.0;`
    - **Assignment statements:** Matches expressions such as `a = b + c;`
    - **Control structures:** Detects `if`, `else`, and `while` blocks using their respective syntax
    - **Print statements:** Identifies `printf()` functions and converts them accordingly

3. **Regex-Based Transformation:** For each pattern matched, a corresponding Promela-equivalent syntax is generated. Some example transformations include:

    - `int a;` → `int a;`
    - `if (a > b)` → `if :: (a > b) ->`
    - `printf("hello");` → `printf("hello");` (preserved as is)

4. **Writing to Output:** The converted Promela code is saved in an output file called `output.pml`. This file is then read and displayed by the GUI for user review.

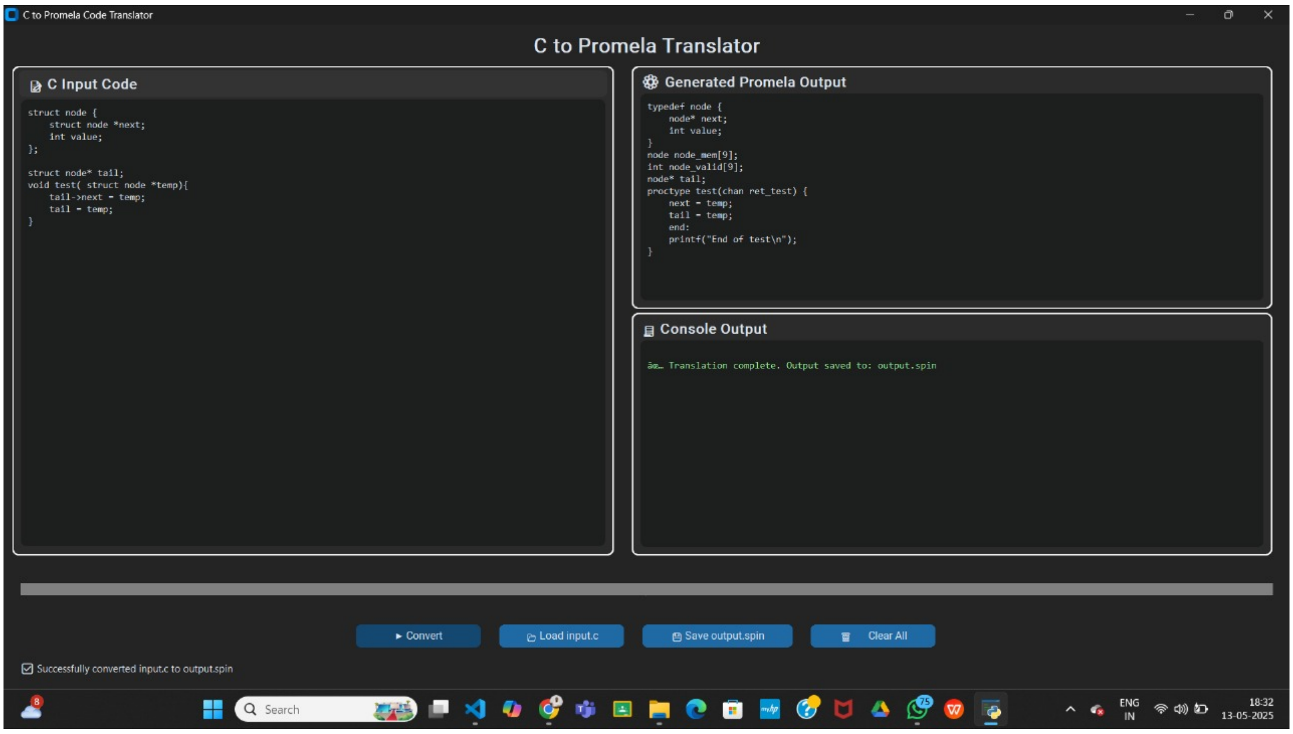
The use of regex provides flexibility and allows for quick identification and translation of code patterns without the need for complex parsers or syntax trees.

3.2. Frontend GUI: User Interface and Execution Controller (`gui.py`)

The graphical user interface (GUI) is built using the `customtkinter` library in Python. It enhances user interaction and provides a seamless experience for code input, translation, and result visualization. The GUI consists of several key components:

- **C Code Input Box:** A text box where users can type or load their C code.
- **Convert Button:** On clicking this, the Python script saves the input code to `input.c`, compiles the C++ backend (`maincode.cpp`) using the `g++` compiler, and executes the binary to generate `output.pml`.
- **Promela Output Box:** This displays the translated Promela code read from `output.pml`.
- **Console Output Box:** Shows the compilation and runtime logs (including any errors).
- **Save/Load Buttons:** Enable users to save the Promela output or load C code from files.

The GUI also includes syntax highlighting for both C and Promela keywords using the built-in tag system of `tkinter`, allowing for better readability.



```
struct node {
    struct node *next;
    int value;
};

struct node* tail;
void test( struct node *temp){
    tail->next = temp;
    tail = temp;
}
```

```
typedef node {
    node* next;
    int value;
}
node node_mem[9];
int node_valid[9];
node* tail;
proctype test(chan ret_test) {
    next = temp;
    tail = temp;
    end:
    printf("End of test\n");
}
```

```
âœ… Translation complete. Output saved to: output.spin
```

```
Successfully converted input.c to output.spin
```

Figure 1: GUI Interface

3.3. Translation Pipeline Summary

The entire system functions as a pipeline where:

1. The user writes or loads a C program in the GUI.
2. The input is saved to a temporary file `input.c`.
3. The backend C++ program reads this file, parses it using regular expressions, and writes the Promela equivalent to `output.pml`.
4. The GUI then reads `output.pml` and displays it in the output text box.
5. Optionally, the user can save the final Promela code for use in model checking tools such as SPIN.

4. Sample Dry Runs:

C Code	Promela Code

`int x = 0;`
`while (x < 5) {`
`    x++;`
`}`
	`int x = 0;`
	`do`
	`    :: (x < 5) ->`
	`        x++;`
	`    :: else -> break;`
	`od;`

Table 1: Comparison between **while loop** of C and Promela

C Code

```
int i, j;
for (i = 0; i < 3; i++) {
    j = 0;
    while (j < 2) {
        arr[i][j] = i + j;
        j++;
    }
}
```

Promela Code

```
i = 0;
do
    :: ( i < 3) ->
        j = 0;
        do
            :: (j < 2) ->
                arr[i][j] = i + j;
                j++;
            :: else -> break;
        od;
    :: else -> break;
od;
```

Table 2: Comparison between **for_while loop** of C and Promela

C Code

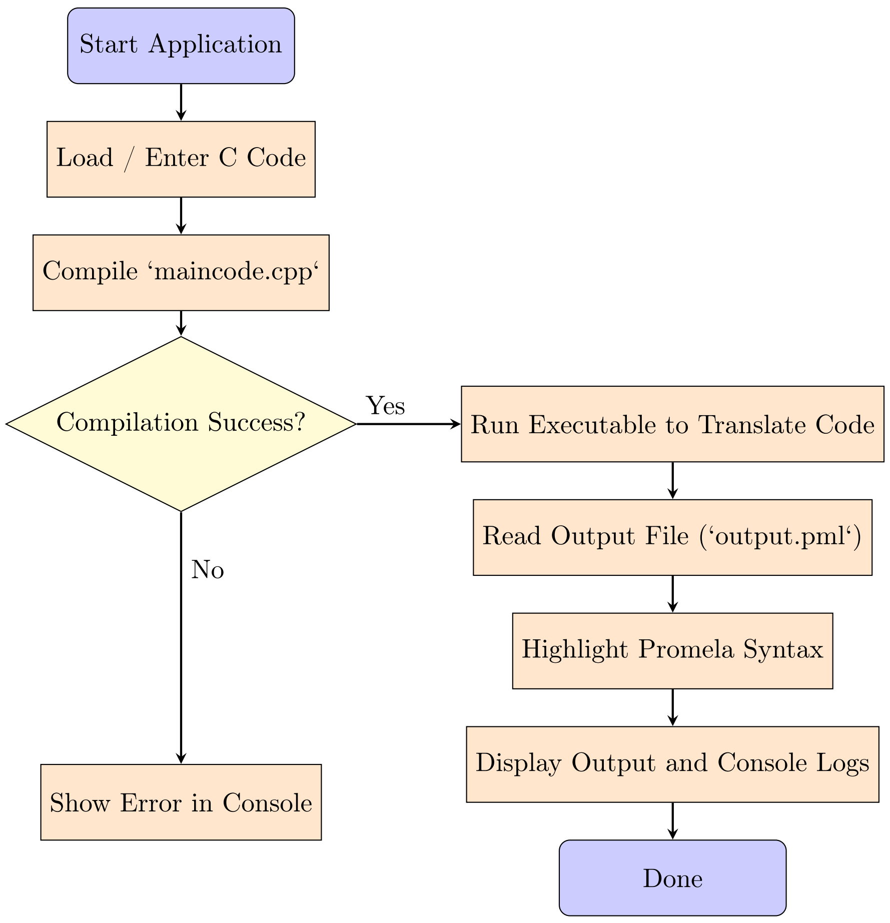
```
int gcd(int x, int y) {
    if (y == 0) return x;
    else return gcd(y, x % y);
}
```

Promela Code

```
proctype gcd(chan ret_gcd; int x; int y) {
    if
        :: (y == 0) ->
            ret_gcd ! x;
            goto end;
        :: else ->
            run gcd(ret_tmp, y, x % y);
            ret_tmp ? tmp;
            ret_gcd ! tmp;
            goto end;
        fi;
        end:
        printf("End of gcd\n");
    }
```

Table 3: Comparison between **GCD Algorithm** in C and Promela

5. Flowchart:



6. Differences Between C and Promela languages:

Aspect	C Language	Promela Language
Purpose	General-purpose programming	Model checking and verification
Execution	Compiled and run as executable	Simulated and verified using SPIN
Data Types	Built-in types: int, float, char, etc.	Limited types: int, bool, chan
Concurrency	Threads or processes (OS-dependent)	Built-in process concept using `proctype`
Control Flow	`if`, `else`, `while`, `for`	`if`, `do`, `fi`, `od` (non-deterministic branching)
Communication	Via variables, memory, or IPC	Via channels using `send (!)`/`receive (?)`
Syntax Style	Curly-brace block structure	Similar structure but more symbolic
Toolchain	Requires compiler (e.g., GCC)	Verified using SPIN tool

Table 4: Differences Between C and Promela Languages

7. Project Summary

This project focuses on the automated translation of C programs into Promela (Process Meta Language) using parsing techniques, with the goal of enabling formal verification through model checking. By analyzing the syntactic and semantic structure of C code, the tool systematically maps key constructs such as data types, control flow (`if`, `while`, `for`), and process-level concurrency into equivalent Promela representations like `proctype`, `chan`, and non-deterministic control blocks (`do`, `od`, `if`, `fi`).
The system utilizes a parsing-driven approach to extract and reinterpret C program logic into a model suitable for verification with the SPIN model checker. This not only ensures correctness in translation but also preserves the behavioral semantics necessary for verifying properties such as deadlocks, safety, and liveness. The automation significantly reduces the manual effort and potential for human error in model transformation, bridging the gap between practical programming in C and formal verification in Promela.

References

[1] `https://www.diva-portal.org/smash/get/diva2:235718/FULLTEXT01.pdf`

[2] Educational reference PDF shared in Google Classroom — helped in understanding Promela syntax and translation logic.

[3] Multiple online sources were referred for proper formatting of Promela code constructs and statement structures (e.g., GitHub repositories, SPIN model checker forums, and documentation blogs).

[4] G. J. Holzmann, *The SPIN Model Checker: Primer and Reference Manual*, Addison-Wesley, 2003.

[5] SPIN Verification Tool, *SPIN Model Checker Website*. `http://spinroot.com/spin/whatispin.html`, Accessed: May 14, 2025.

[6] Tkinter Documentation, *Standard Python GUI Library*. `https://docs.python.org/3/library/tkinter.html`

[7] C++ Reference, *std::regex*. `https://en.cppreference.com/w/cpp/regex`