COMP3931

Individual Project

Project Outline

Summary:
This project aims to develop a Basic Rust Code Logic Analyser specifically designed for the Rust programming language. Drawing inspiration from foundational works in the field, including the 1973 paper "Automatic Program Verification I: A Logical Basis and Its Implementation" and the 1976 paper "Automatic Program Verification V: Verification-Oriented Proof Rules for Arrays, Records, and Pointers". The project aims to integrate classical verification techniques with modern programming paradigms.

By leveraging Hoare logic, the verifier establishes a formal framework for reasoning about program correctness. The implementation seeks to adapt these established principles to the unique features and safety guarantees provided by Rust, thereby enhancing the reliability of software development in a contemporary context.

This analyser aims to provide developers with automated tools to verify the correctness of their Rust programs, ensuring adherence to specified properties and reducing the likelihood of errors.

1 Project Scope

The primary focus of this analyser is to evaluate basic data types and control flow within Rust programs. By concentrating on these fundamental aspects, the analyser aims to provide effective verification without the complexities associated with fully verifying extensive codebases. This targeted approach allows for a more manageable analysis process, ensuring that essential properties of the code can be validated efficiently.

It is important to note that the analyser will not encompass a full termination analysis of programs. Given that the halting problem is undecidable, attempting to ascertain whether arbitrary programs will terminate is outside the scope of this project. Instead, the focus will remain on verifying the correctness of operations within the defined parameters of basic data types and control structures.

Additionally, this analyser will be limited in the types of statements it can interpret, adhering closely to the constraints outlined in the foundational papers on automatic program verification. By imposing these limitations, the analyser can maintain a clear and focused methodology, ensuring that the verification process remains both feasible and effective. This approach allows for meaningful insights into the behaviour of the code while recognising the inherent challenges in broader program verification.

2 Challenges and Difficulities

These are the potential challenges that may arise during development:

1. Rust has a sophisticated type system that includes features like ownership, borrowing, and lifetimes. Accurately modelling these concepts in an analyser can be complex, as they introduce nuances that traditional analysers may not account for.
2. Integrating the analyser with existing Rust tooling and workflows can present challenges. Ensuring compatibility and usability within the Rust ecosystem will require careful design and implementation.
3. As Rust continues to evolve, keeping the analyser updated with the latest language features and paradigms will require ongoing effort and adaptability.
4. Achieving both soundness and completeness in the analyzer is a critical challenge.

3 Implementation Plan

This project will split into these key steps:

1. Understand the original papers, Hoare logic and the Pascal Language.
2. Create a program that will take a Rust program (or snippet) as an input, and parse the program to generate an Abstract Syntax Tree.
3. Generate or take a user's input as the proof goal for the analyser.
4. Generate a complete proof of the program using techniques such as Natural Deduction or Resolution

4 Ethical Problems

Here are some ethical problems that will need to be considered during development:

- Software Reliability and Safety.
- User Trust and Transparency.
- Intellectual Property.
- Bias in Coding Styles.
- Job Displacement.
- Security Vulnerability.
- Responsibility for Errors.
- Accessibility.
- Computational Resources.