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 and Separation 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 a semi-automated tool 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 termination analysis of programs. Given that the halting problem is undecidable, attempting to ascertain whether arbitrary programs will terminate is beyond 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 program 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. The project might be too big to implement in a reasonable time. Some features of Rust may not be verified.

These are the ways to mitigate the protential challenges in the project:

1. Understand the Rust type system and adapt the ideas from Separation Logic. “Unsafe Rust” will not be supported in the analyser.
2. Understand the Rust “cargo” ecosystem and try to integrate the project into cargo. If this is unfeasible, compile the project into a standalone executable.
3. Limit the support of features to a version of Rust, Cargo, and all related Rust Crates. Try to support the latest version of all programs before the deadline of the project.
4. Implement the project by closely following the original automatic program verification papers. Review the project on a fortnightly basis and call off additional implementation if deemed unfeasible.

3 Project Goal and Deliverables

Here is the list of Project Goals and Deliverables:

- Create a summary of the original papers.
- Create a summary of Hoare Logic and Separation Logic.
- Create a summary of the "Hoare Logic and Model Checking" course from the University of Cambridge.
- Compete mock exam questions from the University of Cambridge.
- Create a Rust program parser.
- Generate an AST using the parser.
- Implement the verifier in a programming language.
- Produce a proof of the "quotient remainder program" similar to the original paper.
- Augment new features in the verifier.

The above goals and deliverables are "Specific", "Measurable", "Achievable", and "Relevant". However, due to the complexity of this project, providing a "Time-Bounded" to all the tasks might be difficult. Thus, a bi-weekly review with the supervisor will ensure the project is on track.

In addition to the supervisor meeting, the progress will be recorded on a GitHub "project" page.

Additional support from the School of Mathematics will be available for this project.

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.