RustProof

Requirements Specification Document

Drew Gohman
Matt O'Brien
Bradley Rasmussen
Sami Sahli
Michael Salter
Vincent Schuster
Matthew Slocum

Version: 1.1 **Date: 05/28/2016**

Table of Contents

1. Introduction

1.1 Purpose

This document is intended for both the project sponsors and for project team members to be used as a guide and a mutual agreement of the software to be developed. This document will be used to ensure clarity of expectations and planning, and to verify completeness of the software.

1.2 Scope

The project team shall produce software titled "RustProof," along with a system of Attributes for Rust to be used in conjunction with the software. This software will be a plugin for the Rust Compiler, and will communicate with a Prover. RustProof will receive Attributes, written in the source code by a User, from the Compiler. RustProof will then read and parse these Attributes and translate these Attributes into Verification Conditions to be passed to an automated theorem prover, the Prover. After the theorem prover has finished analyzing the code, RustProof will display the results of this analysis to the user.

The utility of RustProof will serve to allow Rust programmers to generate verification conditions for their code. This ensures that programs written in Rust can be formally verified, thereby reducing the potential of bugs in the code and providing certain guarantees about the behavior of their software.

1.3 Definitions, Acronyms, and Abbreviations.

Attribute - A system of annotated declarations utilized by Rust.
Compiler - The compiler for Rust code, rustc, for which RustProof shall be a plugin.
Prover - The specific automated theorem proving program to be used by the system. Automated theorem proving software takes Verification Conditions and tests their validity, returning an example where a Verification Condition is invalid if one can be found.
Rust - The Rust programming language which the Compiler and RustProof will deal.
RustProof - A Rust Verification Condition generator, the software this document describes the requirements of.
User - A human user who chooses to utilize RustProof. They will insert Attributes into their Rust code to be used by RustProof during compilation.
Verification Conditions - A logical formula derived from Rust code and Attributes associated with that code by a human user, used by a Prover to test validity of a program.

1.4 References

1) Rust style guide: https://aturon.github.io/

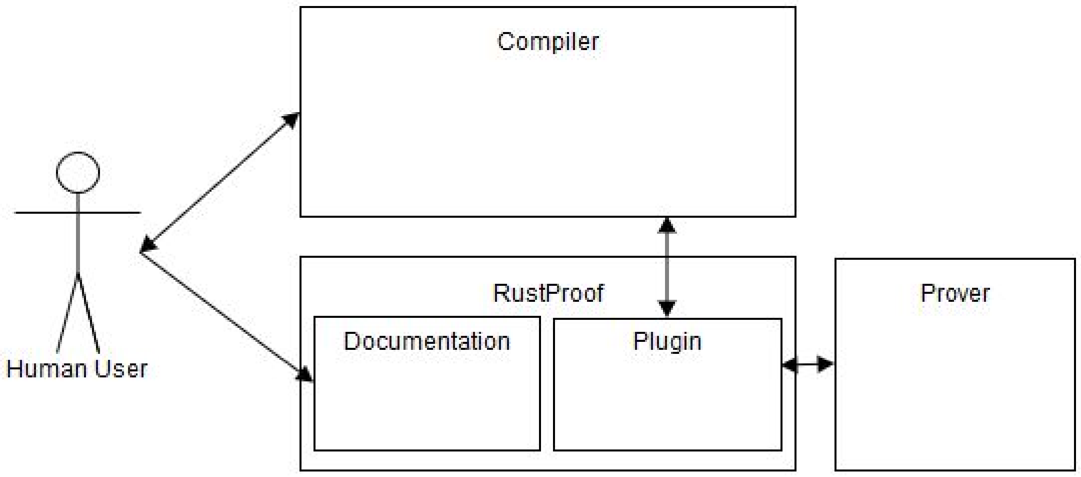
1.5 Overview

This document details the expected uses of the software, what the software will do, what environment it will operate in, and a listing of all applicable requirements it shall or should meet.

2. The Overall Description

2.1 System Interfaces

RustProof will be a plugin for the Compiler and will interact indirectly with Users by parsing code passed to it from the compiler. Verification conditions from the parsed code will be passed to the Prover, and its results returned back to the Compiler, and thus indirectly to the User.



2.1.1 Human Interface

RustProof users shall interact with the software by placing Attributes into their Rust code which will be read by RustProof as input. These Attributes shall use standard Rust formats and should require no new domain knowledge for Users beyond what is used in other Attributes found in Rust; any new information shall be provided in documentation. Because typical users may not have a deep knowledge of English, the documentation shall use simpler language. Any errors or warnings found as a result of RustProof's operation shall be printed to the screen and/or to a log file for the user.

2.1.2 Hardware Interfaces

This software has no direct hardware interfaces.

2.1.3 Software Interfaces

RustProof shall interface as a plugin with the nightly build version of the Compiler. RustProof shall also interface with an established automated theorem prover, the Prover.

2.1.4 Memory Constraints

This software has no memory constraints of note.

2.2 Product Functions

This software shall function as a plugin for the Compiler. This software shall allow the annotation of rust functions with pre- and postconditions as Attributes. These user-defined conditions along with the body of the rust function shall be used to generate semantically equivalent, universally quantified, logical expressions, or Verification Conditions. These Verification Conditions will be tested for contradictions using the Prover, and the results returned to the User through the Compiler.

2.3 User Characteristics

Users are assumed to have at least a working understanding of both Rust and English. Additionally, they are expected to understand how assertions and Attributes are used, but may have no knowledge of automated theorem proving software.

2.5 Assumptions and Dependencies

This document assumes:

- Features of the Rust language used by RustProof will continue to function
- The Compiler will continue to accept plugins
- Attributes submitted in user code will be available to plugins during compilation
- It is possible to determine which Attributes are associated with which code blocks
- A working Prover can be found with a working, convenient interface
- Verification Conditions can be input to the Prover and the results can be output to RustProof
- The Compiler allows plugins to issue warnings and errors
- Rust's system of Attributes will continue to be supported

If any of these assumptions are invalidated, the document and requirements will be changed.

2.6 Apportioning of Requirements

In the minimum viable product, RustProof shall:

- Feature a system of Attributes
- Parse Attributes related to:
  - Integer arithmetic
  - Assertions
  - Immutable functions
- Link Attributes into Verification Conditions
- Output Verification Conditions to Prover
- Output whether Verification Conditions are always true
  - Or, if Verification Conditions are not always true, output counter examples if they exist

In later iterations, RustProof should:

- Parse Attributes related to:
    - Tuples/structs
    - Loops
    - Polymorphic functions without trait bounds

In later iterations, RustProof may:

- Parse Attributes related to:
    - Mutable functions
    - Memory allocation
    - Polymorphic functions with trait bounds
    - Contracts on traits

3. Specific Requirements

3.1 Functions

- The system shall include documentation regarding the Attributes that it supports. (Sponsor meeting, 05/12)
- The system shall accept well-formatted Attributes provided in source code. (Sponsor meeting, 05/12)
    - The system shall only process code for which a precondition, a postcondition, or both have been submitted and shall ignore all else. (Slack conversation, 05/25)
    - The system shall only accept attributes which are attached to functions. (Slack conversation 05/25)
- The system shall construct a set of Verification Conditions from the provided Attributes and code. (Sponsor meeting, 05/12)
- The system shall pass the set of Verification Conditions to the Prover. The individual results from the Prover shall be displayed as Compiler output. (Sponsor meeting, 05/12)
    - If the Verification Conditions cannot be satisfied, the system shall output a counterexample if the Prover provides one. (Sponsor meeting, 05/12)
- The system shall support code containing:
    - Integer arithmetic (Sponsor meeting, 05/12)
    - Assertions (Sponsor meeting, 05/12)
    - Immutable functions (Sponsor meeting, 05/12)
- The system should support code containing:
    - Tuples/structs (Sponsor meeting, 05/12)
    - Loops (Sponsor meeting, 05/12)
    - Polymorphic functions without trait bounds (Sponsor meeting, 05/12)
- The system may support code containing:
    - Mutable functions (Sponsor meeting, 05/12)
    - Memory allocation (Sponsor meeting, 05/12)
    - Polymorphic functions with trait bounds (Sponsor meeting, 05/12)
    - Contracts on traits (Sponsor meeting, 05/12)

3.2 Expected Use Case

Title: Compile code with Attributes
Description: The User runs the Compiler on code that has Attributes for RustProof to parse.
Actors: User (Initiator), Compiler, Prover
Preconditions: The User has code to be compiled with relevant Attributes.
Postconditions: Rustproof returns information from the Prover to the User through the Compiler.
Steps:

1. The User runs the Compiler on the code files they wish to compile.
2. The Compiler processes code until it finds a relevant Attribute, and calls RustProof.
3. RustProof shall parse the Attribute and attempt to extract a valid precondition and postcondition (if either is missing, they are assumed to be "True").
4. RustProof shall generate a weakest precondition for the associated Rust function and the valid postcondition.
5. RustProof shall create a Verification Condition given the valid precondition and the generated weakest precondition.
6. RustProof shall extract the type and names of relevant variables from the Rust function.
7. RustProof shall send the Verification Condition, along with the variable types and names, to the Prover and await results.
8. The Prover should return a result for the Verification Condition; either that it is always true, or that it is sometimes false (possibly with a counterexample).
9. RustProof shall return the results from the Prover to the Compiler, and if counterexamples are provided, RustProof should map them to the associated variables in the function and return that information as well.

Extensions:

3a. RustProof parses the Attribute and it is invalid some way (either from poor formatting, logical errors, or referencing nonexistent variables).
 3a1. RustProof returns an error to the Compiler.
3b. RustProof parses the Attribute and it is valid, but the associated function is of a type that is not supported by RustProof, or contains code of that sort.
 3b1. RustProof returns a warning to the Compiler.
8a. The Prover may hang or become unresponsive, in which case RustProof will also idle. The User will have to stop the program at their discretion.

3.3 Performance Requirements

There are no specific performance requirements for this software.

3.4 Design Constraints

RustProof’s code shall respect the Rust style guidelines (1). RustProof should work on all Rust “Tier 1” platforms.

3.5 Software System Attributes

3.5.1 Security

RustProof shall not save, modify, or delete any User-provided code.

3.5.2 Maintainability

RustProof shall be documented to Rust community standards.

3.5.3 Portability

RustProof should be usable as a plugin for the Compiler on any major system architecture, including Windows, Linux, and MacOS.

3.5.4 Correctness

RustProof shall always produce correct output from a valid input.

3.5.5 Usability

RustProof should be documented using precise, simpler language and require no domain expertise outside of familiarity with Rust, its Attribute system, and the idea of formal verification.

4. Validation and Verification

The development team shall provide sample functions with known validity to test RustProof's features.

5. Change Management Process

Changes to requirements can be submitted by either the sponsors or by the development team, but cannot be approved without being agreed upon by at least five of seven members of the development team and at least one sponsor. This document will then be updated to reflect those changes, and will be approved again.