Rust Education Workshop 2022

Experiences of Teaching Rust and Code Recommendation to Assist Rust Beginners

Hui Xu

School of Computer Science

Fudan University

8/20/2022

Outline

I. **Background**

II. Experiences of Teaching Rust

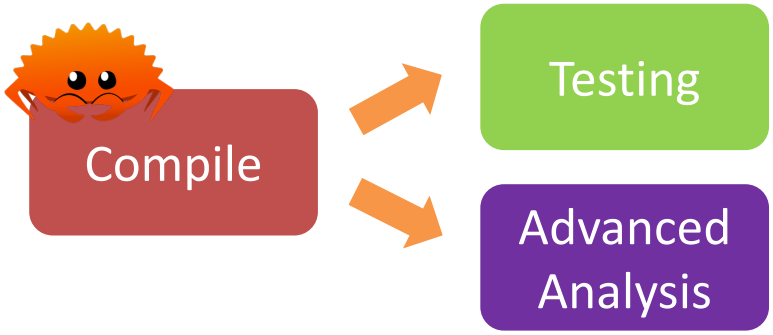
III. Code Recommendation to Assist Beginners

Short Bio

- Tenure-track Associate Professor, Fudan University
- Research Interest: program analysis and software reliability
  - Several publications related to Rust
- Courses I teach at Fudan:
  - COMP 737011 - Memory Safety and Programming Language Design
    - Postgraduate course
    - Design of Rust and the memory-safety issues it aims to address
  - SOFT 130061 - Compiler Theory
    - Undergraduate course
    - A few concepts related to Rust (*e.g.,* type system)

Research Interest In Rust Language

- In software reliability research, we cannot trust developers.
- There are already many papers working on detecting bugs through software testing, static analysis, *etc*.

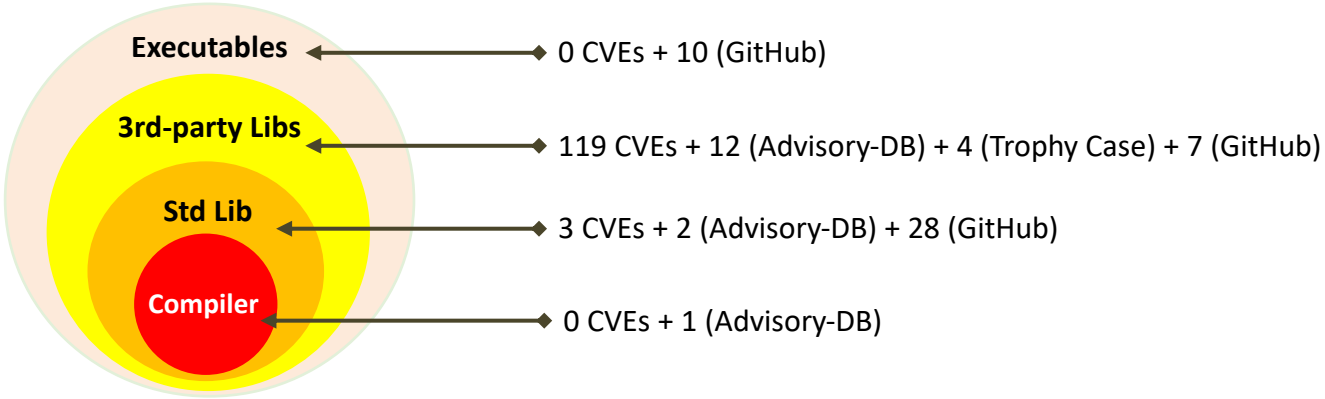


- Rust starts a new trial that aims to prevent critical bugs through language design while still offering adequate control flexibility.
  - It is challenging to balance between security and usability.
- Our research work:
  - Rust bug survey [1]
  - Rust program analysis: RULF [2], SafeDrop [3]

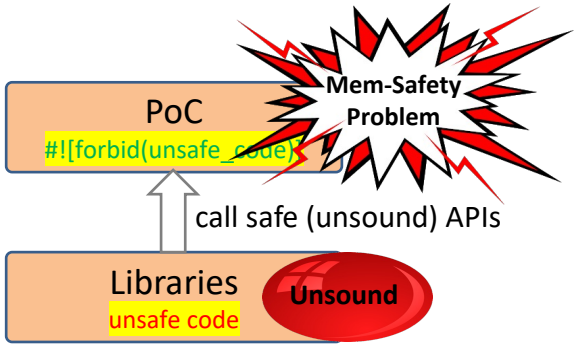
[1] “Memory-safety challenge considered solved? An in-depth study with all Rust CVEs”, TOSEM, 2022.
[2] "RULF: Rust library fuzzing via API dependency graph traversal." ASE, 2021.
[3] “SafeDrop: Detecting memory deallocation bugs of Rust programs via static data-flow analysis.” TOSEM, 2022.

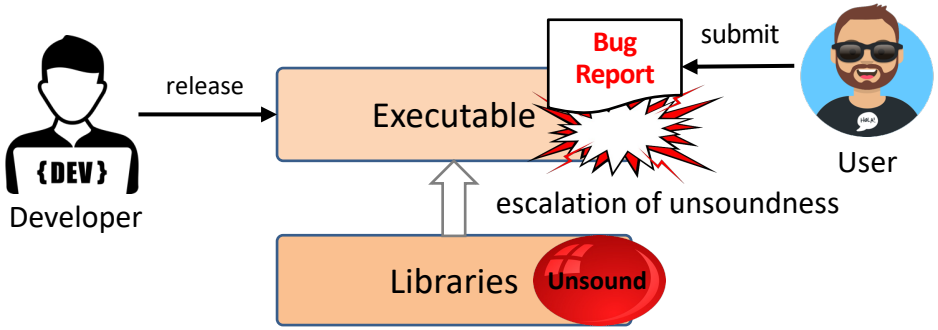
Result of Survey

- Based on a dataset of 185 memory-safety bugs before 2020-12-31



- Rust is effective in memory-safety protection:
  - All these bugs require unsafe code except one compiler bug.
  - Most CVEs are API soundness issues (no CVEs of executables).





[1] “Memory-safety challenge considered solved? An in-depth study with all Rust CVEs”, TOSEM, 2022.

Why Do I Teach Rust?

- Rust is a successful language.
  - My student told me "As long as a Rust program compiles, the executable is likely to work correctly."
  - One senior Rust developer said "I can always feel my skill improvement in using the language."
- A new language with few "legacy features"
  - *e.g.,* C++ intelligent pointers vs Rust ownership + RC
- Appealing features of Rust:
  - Memory-safety guarantee if developers do not use unsafe APIs
  - Powerful type system: type inference, generic, trait bound, *etc.*
  - Exception handling design: Result/Error type, unwinding/abort
  - …

Sample Features I Like

- Variable declaration grammar: type after identifier
  - Much easier to develop an efficient top-down parser (compiler)
  - Compact for type inference: type can be omitted

Rust Code

```
let x:i32 = 1;
let y = 2;
```

C/C++ Code

```
int32_t x = 1;
auto y = 2;
```

- Trait bound: to declare bounded generic parameters
  - Useful for debugging and safety control (Send/Sync)

Rust Code

```
fn max<T:Ord>(x:T,y:T)->T{
    if x > y {x} else {y}
}
```

C/C++ Code

```
template <typename T>
T max(T x, T y) {
    return (x > y) ? x : y;
}
```

Outline

I. Background

II. Experiences of Teaching Rust

III. Code Recommendation to Assist Beginners

COMP 737011 - Memory Safety and Programming Lang. Design

Part1: Foundations of Memory Safety

- Week1: Course Introduction
- Week1: Buffer Overflow
- Week2: Memory Allocation
- Week3: Heap Attack and Protection
- Week4: Memory Exhaustion and Exception Handling
- Week5: Concurrent Memory Access

Part2: Rust Programming Language

- Week6: Rust Ownership-based Memory Management
- Week7: Rust Type System
- Week8: Rust Concurrency
- Week9: Rust Functional Programming
- Week10: Rust Compiler Design
- Week11: Rust Compiler Techniques (by Guest Speaker)

Part3: Advanced Topic for Memory Safety

Due to Covid-19, we have to rearrange the course materials.

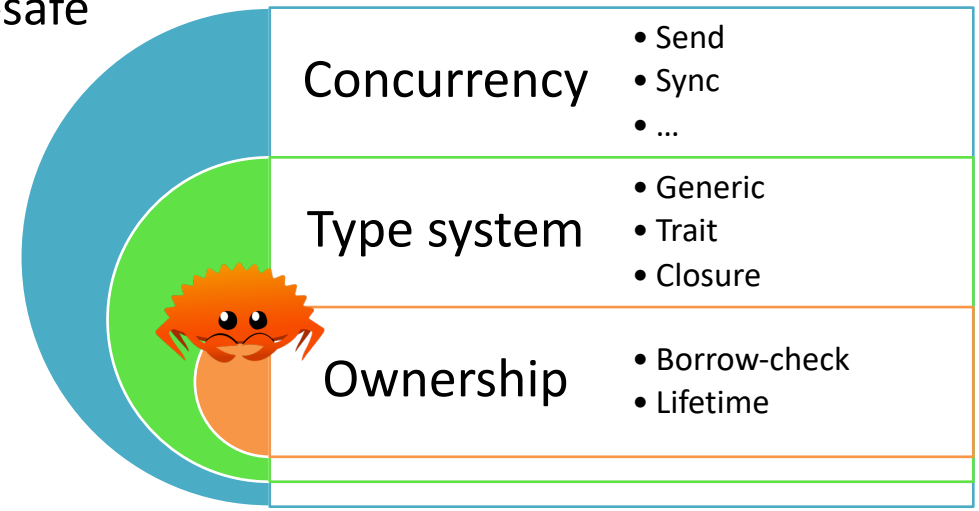
- Week12: Dynamic Analysis of Rust Programs
- Week13: Static Analysis of Rust Programs
- Week14: Presentation (by Students)

- 3 * 45 minutes each weak
  - 2 units for teaching
  - 1 unit for in-class practice
- Grading
  - Attack experiment
  - Coding practice
  - Paper presentation

Course Website: https://hxuhack.github.io/lecture/memsafe

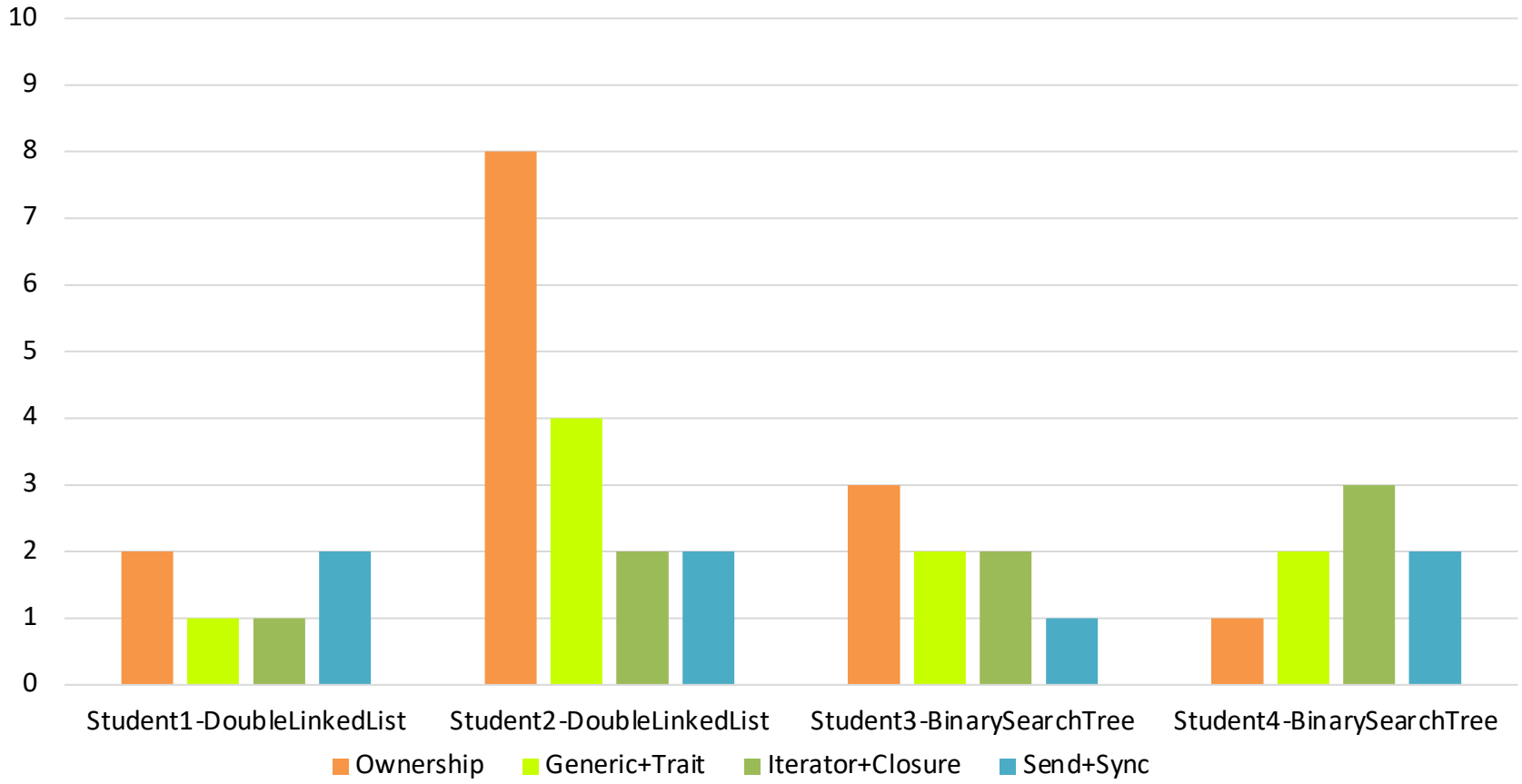
Four Rust Coding Practices

- I: Implement a binary search tree or a double linked list
  - Support insertion, deletion, and search
  - Use safe Rust only
- II: Extend the struct with generic parameters and traits
  - Support generic parameters
  - Implement traits such as Eq and Ord
- III: Implement an iterator for the struct
  - Demonstrate how the filter works with closure
  - Optional feature: collect(), map()
- IV: Rewrite the struct to be thread-safe
  - Implement Sync and Send traits
  - Show the struct is thread-safe

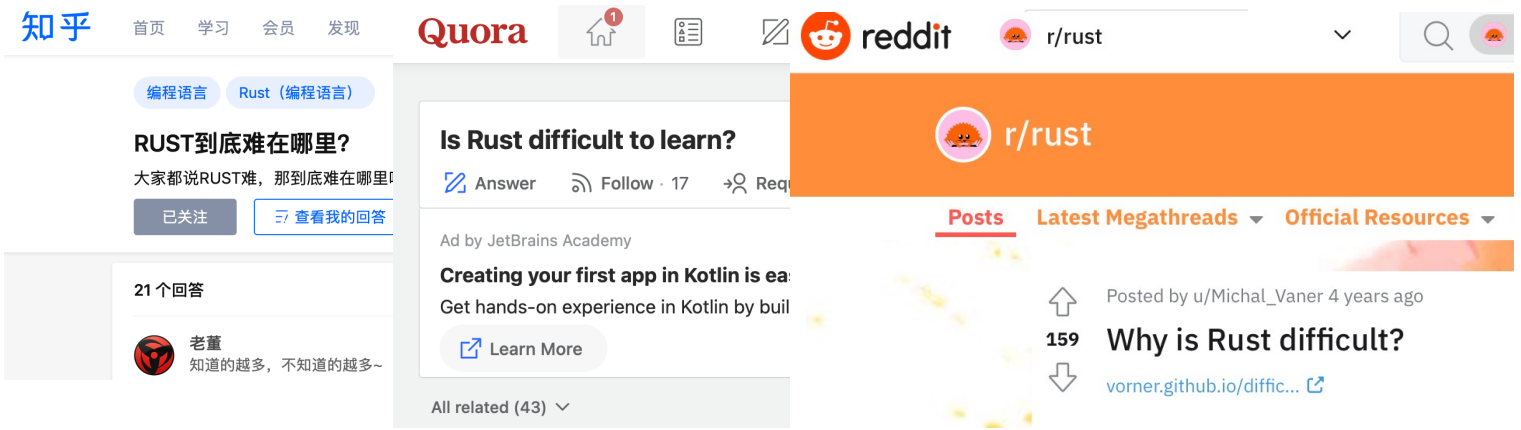


Time Spent on Each Practice

- Most students can finish the assignments in 2 hours.



Is Rust Difficult to Learn?



- Responses from my students:
  - “Unfamiliar with the ownership”
  - “Have much restrictions on developers”
  - “Difficult but interesting. I spent much time combating with the compiler’s borrow check and dereference issues.”
  - “Not that difficult if with C++ background, but I think lifetime is hard.”

My Understanding of Rust's Steep Learning Curve

- Assume a Minimal Rust for beginners? (as I tried in my class)
  - Still not easy to write compliable code but should be manageable
  - Exclusive mutability principle (borrow check)
  - Lifetime mechanism (lifetime inference)
- Many advanced features
  - Bring barriers to reading Rust code written by others
  - Difficult to use these features well
    - *e.g.,* Safe/unsafe, trait bound
    - C/C++ developers may ignore the soundness of their APIs

Outline

I. Background

II. Experiences of Teaching Rust

III. Code Recommendation to Assist Beginners

Code Recommendation (Our Ongoing Project)

- Build a knowledge base that summarizes the common mistakes made by Rust developers
- Make recommendations to developers when coding
- Features can be considered:
  - Compiling errors related to borrow check and lifetime
    - =>Provide better suggestions to fix the bug
  - Unnecessary usage of unsafe code
    - =>Suggest equivalent safe code
  - Other common patterns of bugs
    - =>Warn developers the problem

Example of Replaceable Unsafe Code: MaybeUninit

```
19        static ref BYTE_TO_EMOJI: [String; 256] = {
    -         // SAFETY: safe
    -         let mut m: [MaybeUninit<String>; 256] = unsafe { MaybeUninit::uninit().assume_init() };
20  +         const EMPTY_STRING: String = String::new();
21  +
22  +         let mut m = [EMPTY_STRING; 256];
23            for i in 0..=255u8 {
    -             m[i as usize] = MaybeUninit::new(byte_to_emoji(i));
24  +             m[i as usize] = byte_to_emoji(i);
25            }
    -         unsafe { mem::transmute::<_, [String; 256]>(m) }
26  +         m
27        };
```


https://github.com/bottom-software-foundation/bottom-rs/pull/6/files

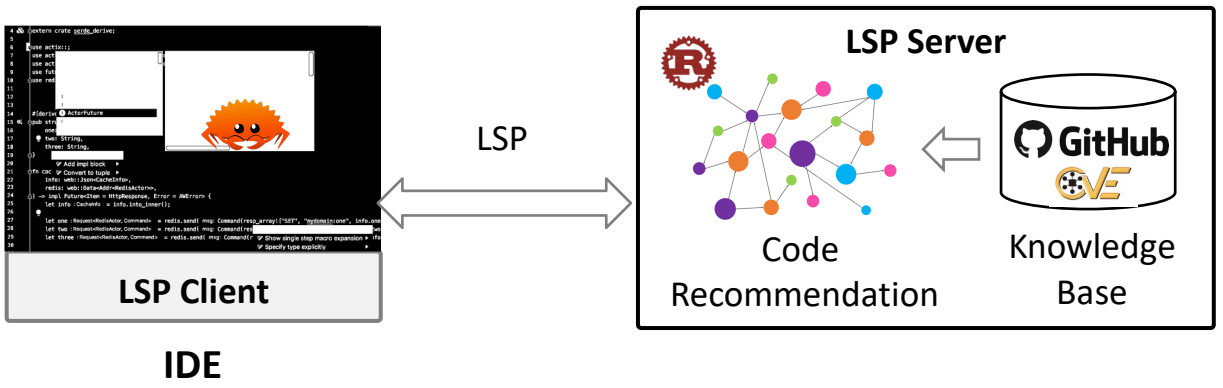
Example of Replaceable Unsafe Code: Raw Pointer

```
    -     fn read_raw_bytes(&mut self, s: &mut [MaybeUninit<u8>]) -> Result<(), String> {
670 +     fn read_raw_bytes_into(&mut self, s: &mut [u8]) -> Result<(), String> {
671               let start = self.position;
    -             let end = start + s.len();
    -             assert!(end <= self.data.len());
    -
    -             // SAFETY: Both `src` and `dst` point to at least `s.len()` elements:
    -             // `src` points to at least `s.len()` elements by above assert, and
    -             // `dst` points to `s.len()` elements by derivation from `s`.
    -             unsafe {
    -                 let src = self.data.as_ptr().add(start);
    -                 let dst = s.as_mut_ptr() as *mut u8;
    -                 ptr::copy_nonoverlapping(src, dst, s.len());
    -             }
    -
    -             self.position = end;
    -
672 +             self.position += s.len();
673 +             s.copy_from_slice(&self.data[start..self.position]);
674               Ok(())
675           }
```

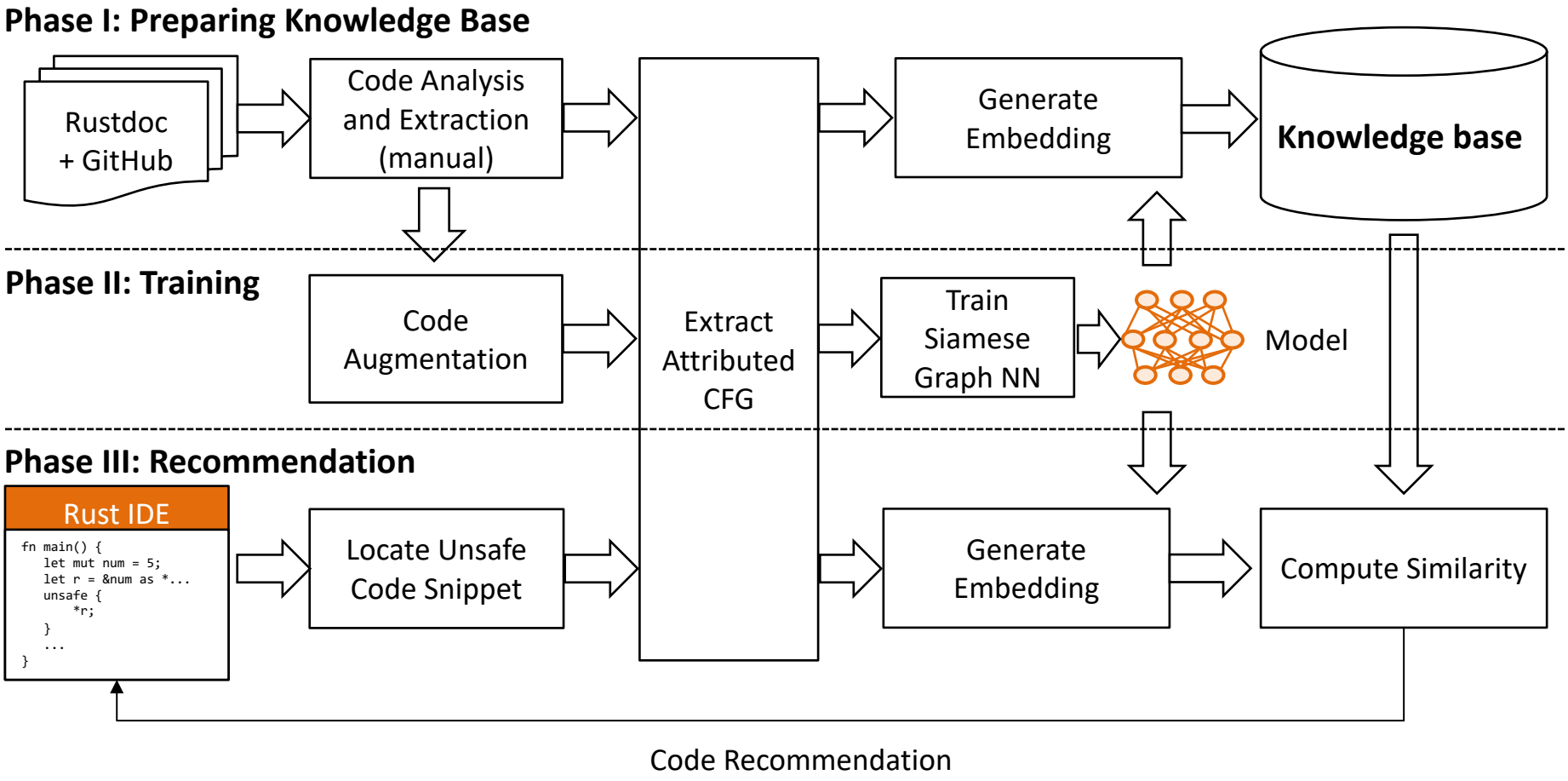

https://github.com/rust-lang/rust/pull/83465/files

Solution Overview

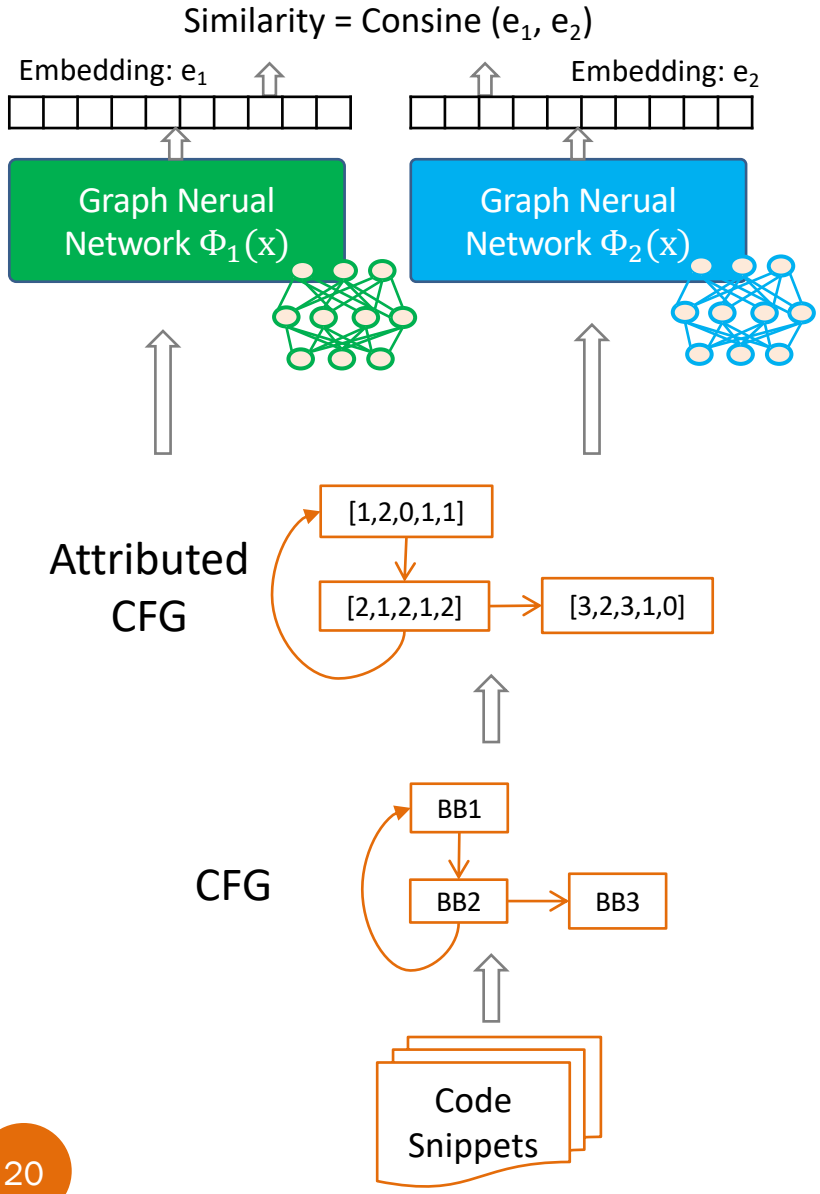
- Based on the language server protocol
  - Or Rust Analyzer (https://rust-analyzer.github.io)
- Advantages:
  - Rely on the power of the server to do complicated analysis tasks, *e.g.,* static analysis, machine learning
  - Perform analysis when coding instead of when compiling
  - One server for several clients
  - Incremental knowledge base



Recommendation Approach



Demonstration of Data Processing



- Siamese graph neural network
  - Similarity between the same code: 1
  - Similarity of different code snippets: 0
- Attributed control-flow graph
  - A directed graph of vectors
  - Each vector represents the features of a basic code block
    - Number of statements
    - Indegree
    - Outdegree
    - …

Conclusion and Takeaways

- Rust is a successful language with many attractive features.
- My experiences of teaching (minimal) Rust is encouraging.
  - Positive feedback based on the performance of my students
- The magic of Rust lies in the soundness requirement of safe APIs.
  - Declarative security
- To assist Rust beginners in writing high quality code, we can summarize common bug patterns and make recommendations.
  - Language server protocol
  - Siamese graph neural network

Thanks for Watching

Q & A