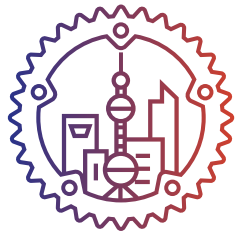
RUST CHINA CONF 2021 - 2022

第二届中国Rust开发者大会

2022.07.31 Online



Rust API可靠性分析与验证

姜剑峰

主题内容

- Rust第三方库API可靠性现状
- 现用方法的局限性
- 基于程序合成+模糊测试的可靠性分析方法

自我简介

- 本（2019）硕（2022）毕业于复旦大学
- 目前在蚂蚁集团安全计算部门开发应用于机密计算的Rust系统软件
- 研究生期间主要从事**Rust测试与验证工具**的研究，本人所在的是国内最早开展**Rust程序分析**相关研究的实验室（https://artisan-lab.github.io）
- 我们关于**Rust库模糊测试**的论文 **RULF: Rust Library Fuzzing via API Dependency Graph Traversal** 获得了软件工程顶级会议ASE2021的ACM杰出论文奖

引言：Rust for Linux

"If the Rust compiler ends up doing hidden allocations, and they then cause panics, then one of the main *points* of Rustification is entirely broken."

—— Linus Torvalds[1]

"There is clearly a need to identify dynamic checks that can never fail and dynamic checks that could potentially fail. "

—— Alastair Reid[2]

[1] https://lwn.net/ml/linux-kernel/CAHk-=wiVY56LzwV_G075NEFwsdf-p7GOTy_cB7-UU9b=49rB1g@mail.gmail.com
[2] https://project-oak.github.io/rust-verification-tools/2021/08/22/rust-on-linux-1.html

Rust静态检查与动态检查

静态检查：

- 基于所有权和生命周期的内存管理模型：内存安全
- 通过trait来确保代码符合某些规范：Send, Sync, Unpin等

动态检查：

- 数组越界
- 整数溢出
- Unicode字符边界

Rust API 可靠性

现有的机制是否足够呢？

- unsafe代码没有破坏内存安全性
- no memory leakage
- panic free

在任何合法使用API的情况下

- 所有静态检查提供的保证都应该被满足（不应该被unsafe所破坏）
- 所有动态检查都不应该被违背（可以被安全的移除），除非panic是一种允许的行为

现有的可靠性分析方法及其局限性

模糊测试(afl.rs, libfuzzer)：分支覆盖率；用例程序的构造

符号执行(klee, angr)：路径爆炸；求解困难

静态分析(MirChecker, Rudra, SafeDrop)：分析特定问题；假阳性

形式化验证(RustBelt)：无法方便的验证第三方库

其他工具（Miri等）.....

API 测试用例合成

```rust
impl Parser {
    fn justfile(mut self) -> Result<Justfile<'a>, CompileError<'a>>
}
```

出错的API[1]

```rust
fn test(input: &str) -> Result<Justfile, CompilerError>{
    let tokens: Vec<Token> = Lexer::lex(input)?;
    let parser: Parser = Parser::new(input, tokens);
    let _justfile: Justfile = parser.justfile()?;
}
```

能够执行到出错API的程序

```rust
fuzz!(|data: &[u8]|) {
    if data.len() < 1 {return;}
    if let Ok(input) = str::from_utf8(data) {
        test(input)?;
    }
}
```

通过模糊测试来寻找到触发崩溃的特定输入

1. https://github.com/casey/just/issues/363

模糊测试简介

```rust
fn toy_fuzz_target(data: &[u8]) {
    if data.len() < 1 {
        return;
    }
    let a = data[0];
    if a > 100 {
        // benign codes
    } else {
        // evil codes
        panic!("error encountered!");
    }
}
```

图.1 一个简单的模糊测试用例程序

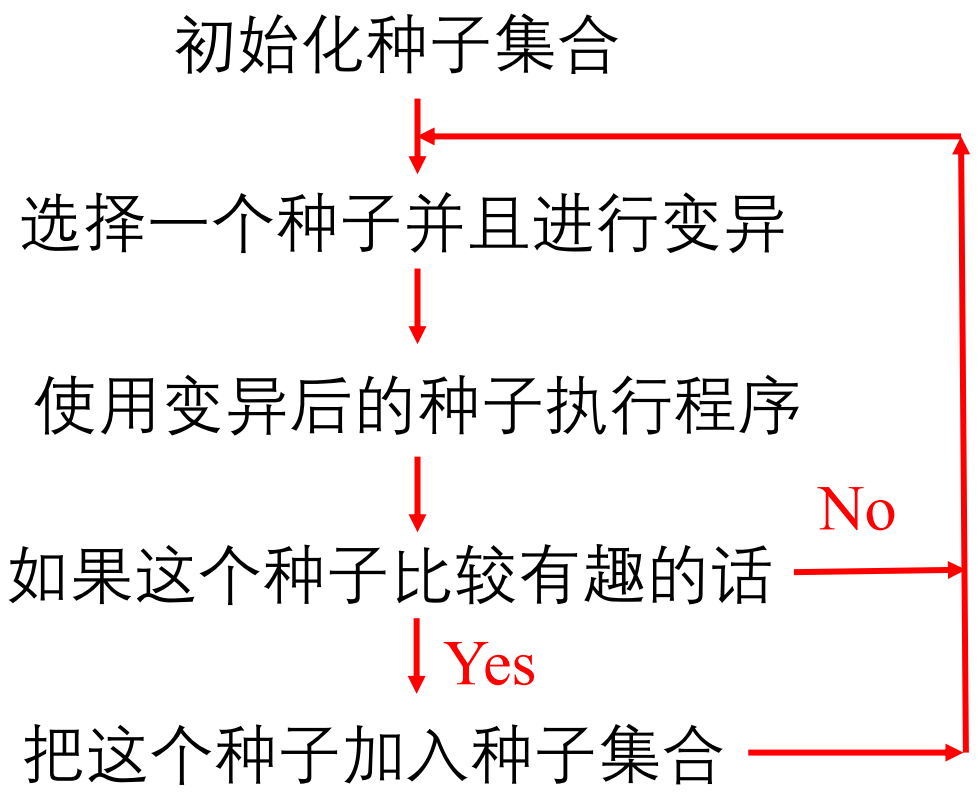
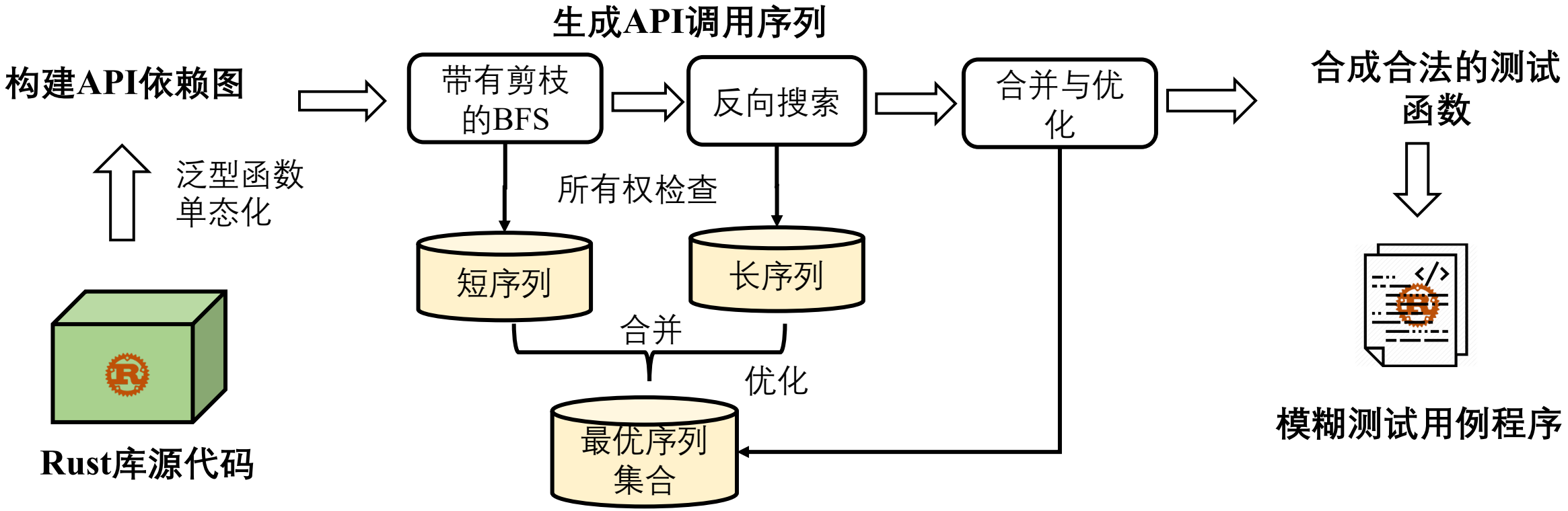


图.2 典型模糊测试工具的工作流程

RULF：模糊测试用例程序生成

开源于https://github.com/Artisan-Lab/RULF



结论以及未来的发展

1. Rust优秀的语言设计避免了大量可能发生的潜在错误
2. 目前Rust程序合成的难点在于unsafe代码，泛型与trait，宏等
3. 保障Rust程序的可靠性仍然需要各种程序分析以及软件测试的手段

Thanks

Rust China Conf 2021-2022 – Online, China