Rethinking Toolchain Design for Rust

Linhai Song[1] and Hao Chen[2]

[1]songlh@ist.psu.edu; [2]chen@ucdavis.edu

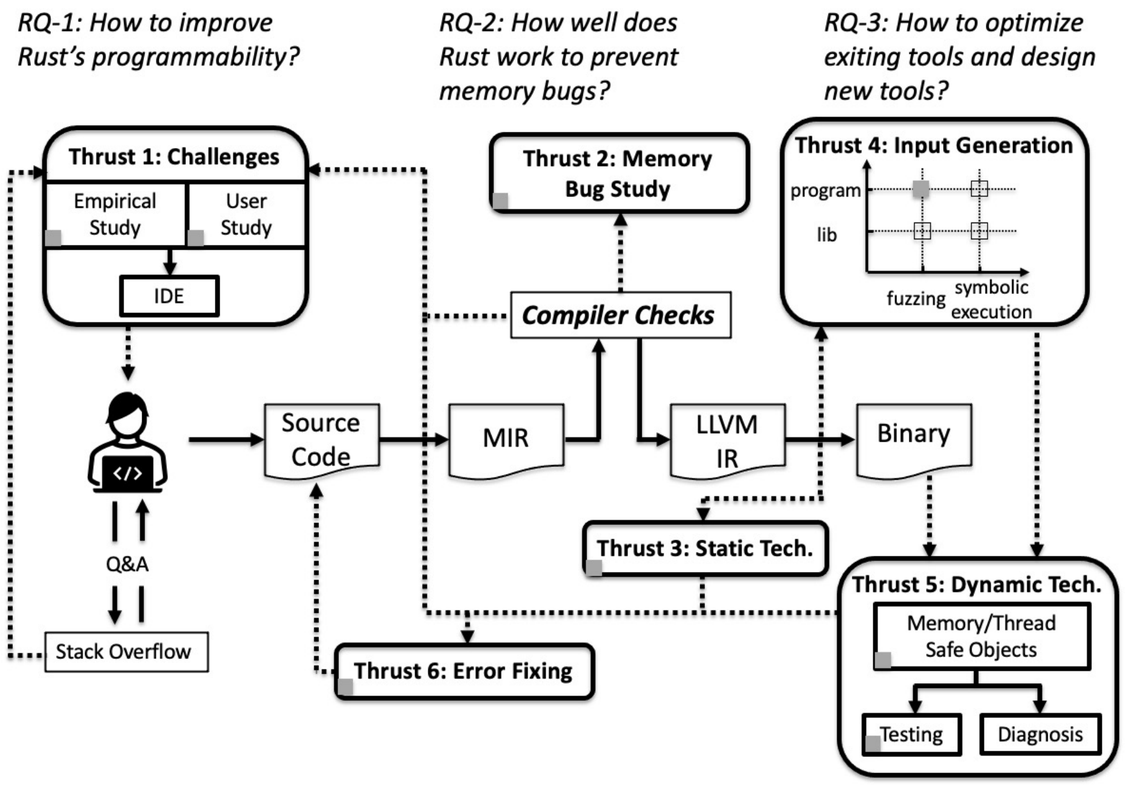
What is *Rust*?

- A young lang. providing both efficiency and safety
- Key innovation is its safety mechanism and rules
  - Centering around ***ownership*** and ***lifetime***
  - Checked against during compilation
- Rust's popularity is increasing dramatically
  - The **most loved** language since 2016
  - Widely adopted to build safety critical software

Rust's Learning and Bug Detection

- Rust has a steep learning curve
  - Safety mechanism is too unique
  - Rust's design philosophy is to reject all suspicious code
- Safety checks can***not*** capture all memory bugs
  - Because of the usage of ***unsafe*** code
  - How to build novel detection techniques for Rust?
  - How to reuse existing techniques (e.g., fuzzing) for Rust?

Rust's Toolchain and Proposed Tasks



Empirical Study on Rust Memory Bugs[1]

- 70 memory bugs from 10 representative apps

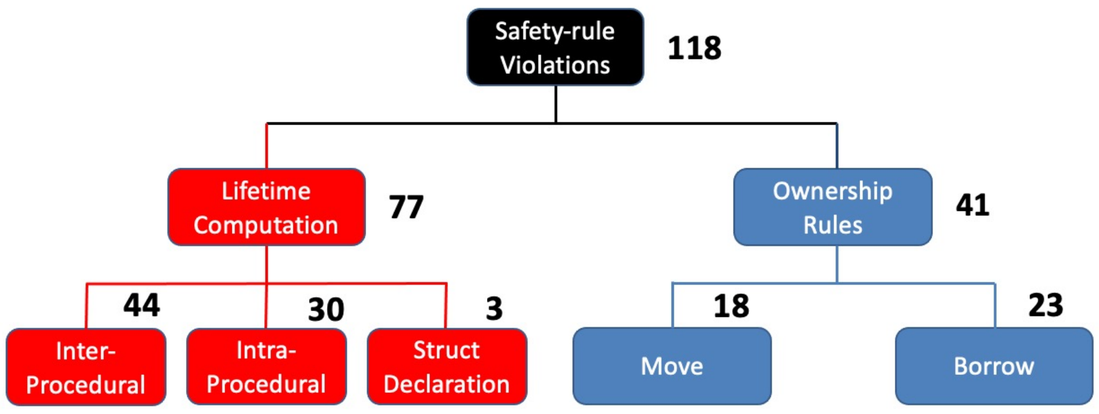
Category	Wrong Access			Lifetime Violation			Total
	Buffer	Null	Uninitialized	IF	UAF	Double Free	
safe	0	0	0	0	1	0	1
unsafe *	4 (1)	12 (4)	0	5 (3)	2 (2)	0	23 (10)
safe -> unsafe *	17 (10)	0	0	1	11 (4)	2 (2)	31 (16)
unsafe -> safe	0	0	7	4	0	4	15

Table 1: Memory Bugs Category. *Buffer: buffer overflow; Null: Null pointer dereferencing; Uninitialized: read uninitialized memory; IF: invalid free; UAF: use after free. *: numbers in () are for bugs whose effects are in interior-unsafe functions.*

Finding 1: Rust's safety mechanisms (in Rust's stable versions) are very effective in preventing memory bugs. All Rust memory bugs involve unsafe code (although many of them also involve safe code).

Rust's Programming Challenges[2,3]

- Study 118 safety-rule violations from Stack Overflow
  - RQ-1: Which Rust safety rules are difficult?
  - RQ-2: When is a safety rule more challenging?
  - RQ-3: How helpful is the Rust compiler?



Finding 2: Rust's safety mechanism may be difficult to apply in concrete usage scenarios.

Finding 3: More lifetime-related questions are asked on Stack Overflow than ownership-related questions, indicating lifetime computation is more challenging in Rust programming.

- Survey 101 real-world Rust programmers
  - To validate the empirical study on Stack Overflow
  - Asking for their previous experience in coding Rust
  - Inspecting how they comprehend safety-rule violations

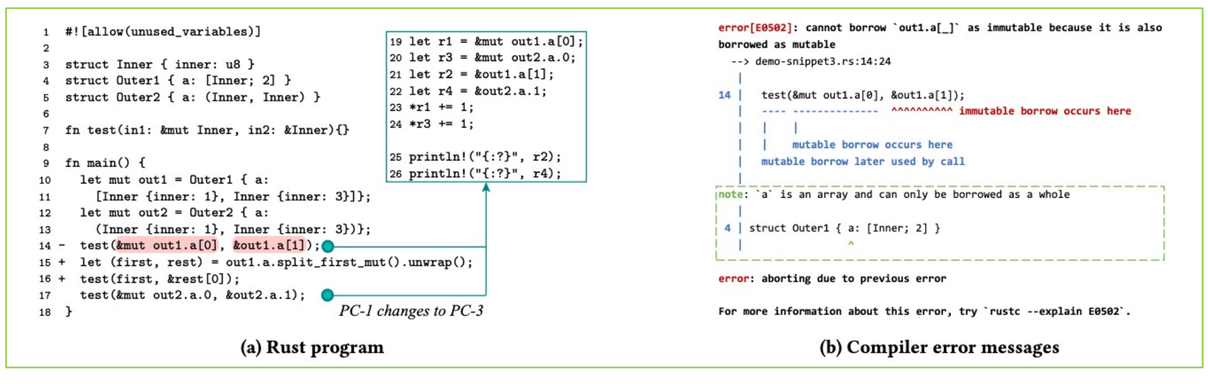


Figure 1: A Rust program and its compiler error. *We replace lines 14 and 18 to create a program variant and add the green-colored rectangle to enhance the error messages.*

Static Bug Detection

Apps	Stars	KLOC	UAF	IF	DL	CL	AV
Servo	19199	320	0	0	0	0	1
Tock	2695	117	0	0	0	0	0
Ethereum	6310	128	0	0	13	14	3
TiKV	8903	237	0	0	0	0	0
Redox	12966	59	5	0	0	0	0
wasmer	8943	75	1	0	12	0	0
substrate	3969	247	0	0	2	0	0
solana	1313	214	0	0	9	8	1
lighthouse	955	102	0	0	9	0	0
rCore	1559	22	0	0	5	3	0
Total	-	-	6	0	50	25	5

Table 2: Detected Bugs. *UAF: use after free; IF: invalid free; DL: double locks; CL: conflicting lock; and AV: atomicity violation.*

NSF Support

- **CNS-1955965**: SaTC: CORE: Small: Collaborative: Understanding and Detecting Memory Bugs in Rust
- **CNS-2145394:** CAREER: Rethinking Toolchain Design for Rust

References

- [1] "Understanding Memory and Thread Safety Practices and Issues in Real-World Rust Programs". Boqin Qin, Yilun Chen, Zeming Yu, Linhai Song, Yiying Zhang. PLDI'2020
- [2] "Learning and Programming Challenges of Rust: A Mixed-Methods Study". Shuofei Zhu, Ziyi Zhang, Boqin Qin, Aiping Xiong, Linhai Song. ICSE'2022.
- [3] "Beyond Bot Detection: Combating Fraudulent Online Survey Taker". Ziyi Zhang, Shuofei Zhu, Jaron Mink, Aiping Xiong, Linhai Song, Gang Wang. WWW'2022.