

Rust's Ownership Model

Florian "Florob" Zeitz

2020-02-05

- 1 Motivation
- 2 Ownership
- 3 Lifetimes
- 4 Rust against our Motivations

1 Motivation

2 Ownership

3 Lifetimes

4 Rust against our Motivations

C++: Realloc

Rust's
Ownership
Model

Florob

Motivation

Ownership

Lifetimes

Rust against
our
Motivations

Questions

```
1  #include <iostream>
2  #include <vector>
3  #include <string>
4
5  int main() {
6      std::vector<std::string> v;
7
8      v.emplace_back("Foo");
9      std::cout << "Capacity: " << v.capacity() << '\n';
10     auto const &x = v[0];
11     v.emplace_back("Bar");
12     std::cout << x << '\n';
13
14     return 0;
15 }
```

C++: Realloc

Rust's
Ownership
Model

Florob

Motivation

Ownership

Lifetimes

Rust against
our

Motivations

Questions

```
1 $ clang++ -std=c++17 -Wall vector.cc -o vector-cc
2 $ ./vector-cc
3 Capacity: 1
4 zsh: segmentation fault (core dumped) ./vector-cc
```

C++: Iterator Invalidation

Rust's
Ownership
Model

Florob

Motivation

Ownership

Lifetimes

Rust against
our
Motivations

Questions

```
1  #include <iostream>
2  #include <string>
3  #include <vector>
4
5  int main() {
6      std::vector<std::string> v = { "F", "O", "O" };
7
8      for (auto const &it : v) {
9          v.push_back(it + it);
10     }
11     for (auto const &it : v) {
12         std::cout << it << '\n';
13     }
14
15     return 0;
16 }
```

C++: Iterator Invalidation

Rust's
Ownership
Model

Florob

Motivation

Ownership

Lifetimes

Rust against
our
Motivations

Questions

```
1 $ clang++ -std=c++17 -Wall iter.cc -o iter-cc
2 $ ./iter-cc
3 F
4 O
5 O
6 FF
7
8
9 $
```

C++: Use After Free

Rust's
Ownership
Model

Florob

Motivation

Ownership

Lifetimes

Rust against
our
Motivations

Questions

```
1  #include <iostream>
2
3  int& f() {
4      int i = 42;
5      return i;
6  }
7
8  void print(int &x) {
9      std::cout << x << '\n';
10 }
11
12 int main() {
13     int &i = f();
14     print(i);
15     return 0;
16 }
```


C++: Use After Free

Rust's
Ownership
Model

Florob

Motivation

Ownership

Lifetimes

Rust against
our
Motivations

Questions

```
1 $ clang++ -Wall after-free.cc -o after-free
2 after-free.cc:5:16: warning: reference to stack memory
   → associated with local variable 'i' returned
   → [-Wreturn-stack-address]
3         return i;
4             ^
5 1 warning generated.
6 $ ./after-free
7 32766
8 $
```

Example: Calculating π

Rust's
Ownership
Model

Florob

Motivation

Ownership

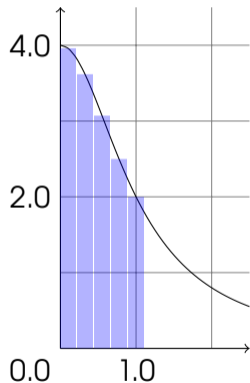
Lifetimes

Rust against
our
Motivations

Questions

$$\begin{aligned}\pi &= 4 \arctan(1) \\ &= \int_0^1 \frac{4}{1+x^2} dx\end{aligned}$$

- Calculate π by Riemann integration
- approximate the area by thin rectangles
- embarrassingly parallel



C++: Data Race

Rust's
Ownership
Model

Florob

Motivation

Ownership

Lifetimes

Rust against
our
Motivations

Questions

```
1  #include <stdint>
2  #include <iostream>
3  #include <thread>
4  #include <vector>
5
6  int main() {
7      constexpr uint64_t NUM_THREADS = 4;
8      constexpr uint64_t NUM_STEPS = 100000;
9      constexpr uint64_t THREAD_STEPS = NUM_STEPS / NUM_THREADS;
10     constexpr double STEP = 1.0 / NUM_STEPS;
11
```

C++: Data Race

Rust's
Ownership
Model

Florob

Motivation

Ownership

Lifetimes

Rust against
our
Motivations

Questions

```
12  double pi = 0;
13
14  std::vector<std::thread> threads;
15
16  for (int i = 0; i < NUM_THREADS; ++i) {
17      uint64_t lower = THREAD_STEPS * i;
18      uint64_t upper = THREAD_STEPS * (i+1);
19      threads.emplace_back([=, &pi]() {
20          for (uint64_t j = lower; j < upper; ++j) {
21              double x = (j + 0.5) * STEP;
22              pi += 4.0 / (1.0 + x*x) * STEP;
23          }
24      });
25 }
```

C++: Data Race

Rust's
Ownership
Model

Florob

Motivation

Ownership

Lifetimes

Rust against
our
Motivations

Questions

26

27

```
for (auto &t : threads) t.join();
```

28

29

```
std::cout.precision(10);
```

30

```
std::cout << "Pi = " << pi << '\n';
```

31

32

```
return 0;
```

33

```
}
```

C++: Data Race

Rust's
Ownership
Model

Florob

Motivation

Ownership

Lifetimes

Rust against
our
Motivations

Questions

```
1 $ clang++ -std=c++17 -lpthread -Wall pi.cc -o pi-cc
2 $ ./pi-cc
3 Pi = 1.156130797
4 $ ./pi-cc
5 Pi = 1.099799814
```

- classical data race
- Thread A liest `pi = 0.1423`
- Thread B liest `pi = 0.1423`
- Thread A schreibt `pi = 0.7609`
- Thread B schreibt `pi = 0.5768`
- `pi = 0.5768`, Thread As Berechnung ist verloren

Observation

Rust's
Ownership
Model

Florob

Motivation

Ownership

Lifetimes

Rust against
our
Motivations

Questions

Problems arise when combining:

Mutability

+

Aliasing

```
v.emplace_back(...)  
v.push_back(...)
```

```
auto const &x = v[0]  
auto const &it : v
```

1 Motivation

2 Ownership

3 Lifetimes

4 Rust against our Motivations

You might have heard

Rust's
Ownership
Model

Florob

Motivation

Ownership

Lifetimes

Rust against
our
Motivations

Questions

- Rust has a linear type system
i.e. each value may only be used once
- Rust has an affine type system
i.e. each value may be used at most once
- Rust is related to quantitative type theory
i.e. values track how often they may be used as type state

Ownership: Bindings

Rust's
Ownership
Model

Florob

Motivation

Ownership

Lifetimes

Rust against
our
Motivations

Questions

```
1  struct Crop;  
2  
3  fn main() {  
4      let c = Crop;  
5  
6      let _grinder1 = c; // moves c to _grinder1  
7  
8      let _grinder2 = c; // error: use of moved value: `c`  
9  }
```

Ownership: Functions

Rust's
Ownership
Model

Florob

Motivation

Ownership

Lifetimes

Rust against
our
Motivations

Questions

```
1  struct Crop;  
2  struct Flour;  
3  
4  fn grind(_c: Crop) -> Flour {  
5      Flour  
6      // _c is freed here  
7  }  
8  
9  fn main() {  
10     let c = Crop;  
11  
12     grind(c); // c moves into `grind()`  
13     grind(c); // error: use of moved value: `c`  
14 }
```

Returning Ownership

```
1  struct Book { page: u32 }
2
3  fn read(b: Book) -> Book {
4      println!("I read page {}", b.page);
5      b
6  }
7
8  fn main() {
9      let b = Book { page: 1 };
10     let b1 = read(b); // b moves into `read()`
11     // let b2 = read(b); // error: use of moved value: `b`
12     let _b2 = read(b1);
13 }
```

Returning Ownership (Mutable)

Rust's
Ownership
Model

Florob

Motivation

Ownership

Lifetimes

Rust against
our
Motivations

Questions

```
8  fn turn_page(mut b: Book) -> Book {
9      b.page += 1;
10     b
11 }
12
13 fn main() {
14     let b = Book { page: 1 };
15
16     let b1 = read(b);
17     let b2 = turn_page(b1);
18     let _b3 = read(b2);
19 }
```

Shared Borrow

Rust's
Ownership
Model

Florob

Motivation

Ownership

Lifetimes

Rust against
our

Motivations

Questions

```
1 struct Book { page: u32 }
2
3 fn read(b: &Book) {
4     println!("I read page {}", b.page);
5 }
6
7 fn main() {
8     let b = Book { page: 1 };
9     let l = &b;
10
11     read(&b);
12     read(l);
13     read(&b);
14 }
```

Mutable Borrow

Rust's
Ownership
Model

Florob

Motivation

Ownership

Lifetimes

Rust against
our
Motivations

Questions

```
7  fn turn_page(b: &mut Book) { b.page += 1; }
8
9  fn main() {
10     let mut b = Book { page: 1 };
11
12     read(&b);
13     turn_page(&mut b);
14     read(&b);
15
16     let l = &b;
17     // turn_page(&mut b); // error: cannot borrow `b` as
18                          // mutable because it is also
19                          // borrowed as immutable
20     read(l);
21 }
```

Intermission: Copy Types

Rust's
Ownership
Model

Florob

Motivation

Ownership

Lifetimes

Rust against
our
Motivations

Questions

```
1  struct Dress;  
2  #[derive(Copy, Clone)]  
3  struct Mp3;  
4  
5  fn main() {  
6      let shop_dress = Dress;  
7      let _your_dress = shop_dress;  
8      let _their_dress = shop_dress; // error:  
9                                     // use of moved value:  
10                                    // `shop_dress`  
11  
12     let shop_mp3 = Mp3;  
13     let _your_mp3 = shop_mp3;  
14     let _their_mp3 = shop_mp3; // This is fine  
15 }
```


1 Motivation

2 Ownership

3 Lifetimes

4 Rust against our Motivations

Basic lifetimes

Rust's
Ownership
Model

Florob

Motivation

Ownership

Lifetimes

Rust against
our
Motivations

Questions

```
1  fn main() {
2      let r;
3
4      {
5          let x = 5;
6          r = &x; // error: `x` does not live long enough
7      }
8
9      println!("r: {}", r);
10 }
```

Basic lifetimes

```
1 fn main() {
2     let r; // -----+-- 'a
3           //      |
4     {     //      |
5         let x = 5; // -+-- 'b |
6         r = &x;    //  |   |
7     }         // -+   |
8           //      |
9     println!("r: {}", r); //      |
10 }           // -----+
```

- x must live until the last use of r

Returning References

Rust's
Ownership
Model

Florob

Motivation

Ownership

Lifetimes

Rust against
our
Motivations

Questions

```
1 struct Page;  
2 struct Book {  
3     page: u32,  
4     content: Vec<Page>,  
5 }  
6  
7 fn get_page<'a>(b: &'a Book, page: usize) -> &'a Page {  
8     &b.content [page]  
9 }
```

Multiple Input Lifetimes

Rust's
Ownership
Model

Florob

Motivation

Ownership

Lifetimes

Rust against
our
Motivations

Questions

```
1 struct Book { page: u32 }
2
3 fn longer<'a>(a: &'a Book, b: &'a Book) -> &'a Book {
4     if a.page > b.page {
5         a
6     } else {
7         b
8     }
9 }
```

Lifetime Elision

Rust's
Ownership
Model

Florob

Motivation

Ownership

Lifetimes

Rust against
our
Motivations

Questions

```
1 struct Page;  
2 struct Book {  
3     page: u32,  
4     content: Vec<Page>,  
5 }  
6  
7 fn get_page(b: &Book, page: usize) -> &Page {  
8     &b.content [page]  
9 }
```

- if there is only one input lifetime all outputs get it
- in methods all outputs get `self`'s lifetime

1 Motivation

2 Ownership

3 Lifetimes

4 Rust against our Motivations

Rust: Realloc

Rust's
Ownership
Model

Florob

Motivation

Ownership

Lifetimes

Rust against
our
Motivations

Questions

```
1 fn main() {  
2     let mut v: Vec<String> = Vec::new();  
3  
4     v.push("Foo".to_string());  
5     let x: &String = &v[0];  
6     v.push("Bar".to_string());  
7  
8     println!("{}", x);  
9 }
```


Rust: Realloc

Rust's
Ownership
Model

Florob

Motivation

Ownership

Lifetimes

Rust against
our
Motivations

Questions

```
1 $ rustc vector.rs
2 error[E0502]: cannot borrow `v` as mutable because it is also
   ↳ borrowed as immutable
3   --> vector.rs:6:5
4   |
5   5 |     let x: &String = &v[0];
6   |                               - immutable borrow occurs here
7   6 |     v.push("Bar".to_string());
8   |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ mutable borrow occurs here
9   7 |
10  8 |     println!("{}", x);
11  |                               - immutable borrow later used here
12
13 error: aborting due to previous error
14
15 For more information about this error, try `rustc --explain E0502`.
```

Rust: Iterator Invalidation

Rust's
Ownership
Model

Florob

Motivation

Ownership

Lifetimes

Rust against
our
Motivations

Questions

```
1  fn main() {
2      let mut v: Vec<String> = vec!["F".into(), "o".into(),
3                                     "o".into()];
4
5      for it in &v {
6          v.push(it.clone() + &it);
7      }
8      for it in &v {
9          println!("{}", it);
10     }
11 }
```

Rust: Iterator Invalidation

Rust's
Ownership
Model

Florob

Motivation

Ownership

Lifetimes

Rust against
our
Motivations

Questions

```
1 $ rustc iter.rs
2 error[E0502]: cannot borrow `v` as mutable because it is also
   ↳ borrowed as immutable
3   --> iter.rs:6:9
4   |
5   5 |         for it in &v {
6   |             --
7   |             |
8   |             immutable borrow occurs here
9   |             immutable borrow later used here
10  6 |             v.push(it.clone() + &it);
11  |             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ mutable borrow occurs here
12
13 error: aborting due to previous error
14
15 For more information about this error, try `rustc --explain E0502`.
```

Calculating π

Rust's
Ownership
Model

Florb

Motivation

Ownership

Lifetimes

Rust against
our
Motivations

Questions

Lifecoding

Thank you for your attention.
Any questions?



<https://babelmonkeys.de/~florob/talks/RC-2020-02-05-rust-ownership.pdf>