Unsafe Rust and Miri

Ralf Jung (ETH Zurich)

Rust Zurich Meetup, June 2023

Rust is a memory-safe language.

Rust is a memory-safe language.

The compiler guarantees that Rust programs do not

- Access memory out-of-bounds
- Access memory after it has been freed
- Execute the “wrong” code
- Have data races

Rust is a memory-safe language.

The compiler guarantees that Rust programs do not

- Access memory out-of-bounds
- Access memory after it has been freed
- Execute the “wrong” code
- Have data races

Except...

unsafe code

```
fn evil() { unsafe {
  let x = 0i32;
  let ptr = ptr::from_ref(&x);
  ptr.offset(1).read(); // uh-oh...
} }
```

In unsafe Rust, memory safety becomes the programmer's responsibility.

The Contract

There is a contract between programmer and compiler, with obligation for both sides

- Compiler's obligations:
  Generate a binary that matches the Rust source code
- Programmer's obligations:
  Write a program that follows "the rules"

If you break that contract, anything can happen!

If you break that contract, anything can happen!

We call this Undefined Behavior.

But what about “soundness”?

Soundness is the property of an API to avoid Undefined Behavior when called from safe code.*

```
pub fn call_me(x: &bool) { unsafe {
  let ptr = ptr::from_ref(x).cast::<u8>();
  if ptr.read() == 2 {
    hint::unreachable_unchecked();
  }
} }
```

But what about “soundness”?

Soundness is the property of an API to avoid Undefined Behavior when called from safe code.*

```
pub fn call_me(x: &bool) { unsafe {
  let ptr = ptr::from_ref(x).cast::<u8>();
  if ptr.read() == 2 {
    hint::unreachable_unchecked();
  }
} }
```

What about the raw pointer version?

Key terminology

- Undefined Behavior: A program that violates "the rules" during its execution
- Unsound Code: A library that can be used by safe code to cause Undefined Behavior

Common misconceptions

- “My code is fine, I tested it.”

Common misconceptions

- “My code is fine, I tested it.”
- “It’s what the hardware does”

Common misconceptions

- “My code is fine, I tested it.”
- “It’s what the hardware does”
- “This code was fine with a previous compiler, but now it broke; this is the compiler’s fault.”

Common misconceptions

- “My code is fine, I tested it.”
- “It’s what the hardware does”
- “This code was fine with a previous compiler, but now it broke; this is the compiler’s fault.”
- “That evil compiler should tell me about the UB instead of exploiting it!”

Common misconceptions

- “My code is fine, I tested it.”
- “It’s what the hardware does”
- “This code was fine with a previous compiler, but now it broke; this is the compiler’s fault.”
- “That evil compiler should tell me about the UB instead of exploiting it!”
- “This is terrible, I will go back to C / use inline assembly.”

Common misconceptions

- “My code is fine, I tested it.”
- “It’s what the hardware does”
- “This code was fine with a previous compiler, but now it broke; this is the compiler’s fault.”
- “That evil compiler should tell me about the UB instead of exploiting it!”
- “This is terrible, I will go back to C / use inline assembly.”
- “I disabled optimizations so UB cannot be a problem.”

Common misconceptions

- “My code is fine, I tested it.”
- “It’s what the hardware does”
- “This code was fine with a previous compiler, but now it broke; this is the compiler’s fault.”
- “That evil compiler should tell me about the UB instead of exploiting it!”
- “This is terrible, I will go back to C / use inline assembly.”
- “I disabled optimizations so UB cannot be a problem.”
- “It’s fine if I put `unsafe` around it.”

Knowing “the rules” is absolutely crucial when writing unsafe code!

But knowing all the rules is hard...

Thankfully, there is a tool to help:

Miri

But knowing all the rules is hard...

Thankfully, there is a tool to help:

Miri

- Miri can detect Undefined Behavior in a program or test suite
- Miri cannot ensure a library is sound!

But knowing all the rules is hard...

Thankfully, there is a tool to help:

Miri

- Miri can detect Undefined Behavior in a program or test suite
- Miri cannot ensure a library is sound!

Caveat: Miri only supports pure Rust code.
No C FFI.

But knowing all the rules is hard…

Thankfully, there is a tool to help:

Miri

Try it on your crate:

```
rustup toolchain install nightly --component miri
cargo +nightly miri test
```

program or test suite

- Miri cannot ensure a library is sound!

Caveat: Miri only supports pure Rust code.
No C FFI.

Miri cannot ensure a library is sound!

So, it is still worth knowing “the rules”.

Examples of Undefined Behavior

- Out-of-bounds access
- Use-after-free
- Out-of-bounds pointer arithmetic
- Insufficient alignment (1) (2)
- Invalid value (1) (2) (3)
- Violation of reference aliasing rules (1) (2) (3)
- Data race

Examples of Undefined Behavior

- Out-of-bounds access
- Use-after-free
- Violation of reference aliasing rules (1) (2) (3)
- Data race

For a more complete list, see the reference. However, specifying this in full precision is still work-in-progress.

`unsafe` library functions always document their requirements, make sure to take good notice of that.

When writing unsafe Rust:

Remember not to break “the rules”!

Further reading

- With Undefined Behavior, Anything is Possible
- Undefined Behavior deserves a better reputation
- Why even unused data needs to be valid
- "What The Hardware Does" is not What Your Program Does: Uninitialized Memory