Compile-time Deadlock Detection in Rust using Petri Nets

Horacio Lisdero Scaffino

Facultad de Ingeniería
Universidad de Buenos Aires

June 30, 2023

Agenda

1. Introduction
2. Rust
   - What is Rust?
   - How does it look like?
   - Why Rust?
3. Petri nets
   - Examples
4. Translation
   - MIR
   - Modelling threads
   - Modelling mutexes
   - Modelling condition variables

Agenda

Agenda

Agenda

What is Rust?

Rust is a multi-paradigm, general-purpose programming language that aims to provide developers with a safe and efficient way to write low-level code.

What is Rust?

Rust is a multi-paradigm, general-purpose programming language that aims to provide developers with a safe and efficient way to write low-level code.

- Memory-safe
- Compiled to machine code, no runtime needed
- High-level simplicity
- Low-level performance (on the same level as C or C++)

Brief timeline of Rust

- 2007 Started as a side project by Graydon Hoare, a programmer at Mozilla
- 2009 Mozilla officially started sponsoring the project
- 2015 First stable version 1.0
- 2016 Mozilla releases Servo, a browser engine built with Rust
- 2019 `async/await` support stabilized
- 2021 The Rust Foundation is founded by AWS, Huawei, Google, Microsoft, and Mozilla
- 2021 The Android Open Source Project encourages the use of Rust for the SO components below the ART
- 2022 The Linux kernel adds support for Rust alongside C
- 2023 8 years in a row the most loved programming language in the Stack Overflow Developer Survey

Memory safety

It achieves memory safety without using a garbage collector or reference counting. Instead, it uses the concept of **ownership** and **borrowing**.

Memory safety

It achieves memory safety without using a garbage collector or reference counting. Instead, it uses the concept of **ownership** and **borrowing**.

It prevents a wide variety of error classes at compile-time:

- Double free
- Use after free
- Dangling pointers
- Data races
- Passing non-thread-safe variables

If a violation of the compiler rules is found, the program will simply not compile.

Agenda

Immutability by default

```rust
fn main() {
  let x = 1;
  x = x + 1;
}
```

```
error[E0384]: cannot assign twice to immutable variable `x`
 --> src/main.rs:3:5
  |
2 |     let x = 1;
  |         -
  |         |
  |         first assignment to `x`
  |         help: consider making this binding mutable: `mut x`
3 |     x = x + 1;
  |     ^^^^^^^^^ cannot assign twice to immutable variable
```

Move semantics by default

Each value has only one owner. If a variable is passed to another function or scope, the owner of the value changes.

```rust
fn main() {
  let name = String::from("Alice");
  print_name(name);
  println!("The name is: {}", name); // Compilation error
}

fn print_name(name: String) {
    println!("Name: {}", name);
}
```

Agenda

Agenda

Agenda

Agenda

Agenda

Agenda

Agenda

Agenda

Bibliography