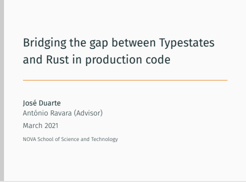
Bridging the gap between Typestates and Rust in production code

José Duarte
António Ravara (Advisor)
March 2021
NOVA School of Science and Technology

Bridging the gap between Typestates and Rust in production code



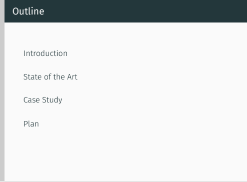
Bridging the gap between Typestates and Rust in production code

José Duarte
António Ravara (Advisor)
March 2021
NOVA School of Science and Technology

Hello everyone! My name is José Duarte and today I will be talking about using typestates in Rust. I'll present:

- A brief definition of typestates.
- Why they are useful.
- And finally I'll discuss their relationship with Rust and my proposal to integrate them in the ecosystem.

Outline


Outline

Introduction

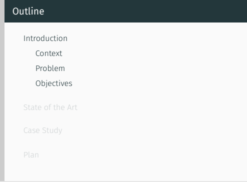
State of the Art

Case Study

Plan

Durante a apresentação irei introduzir o tema, rever sumáriamente o estado da arte, apresentar a proposta de trabalho e por fim rever o plano de trabalho da mesma.

Outline



Outline

Introduction

Context

Problem

Objectives

State of the Art

Case Study

Plan

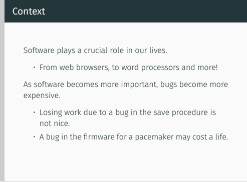
Context

Software plays a crucial role in our lives.

- From web browsers, to word processors and more!

As software becomes more important, bugs become more expensive.

- Losing work due to a bug in the save procedure is not nice.
- A bug in the firmware for a pacemaker may cost a life.


Context

Software plays a crucial role in our lives.

- From web browsers, to word processors and more!

As software becomes more important, bugs become more expensive.

- Losing work due to a bug in the save procedure is not nice.
- A bug in the firmware for a pacemaker may cost a life.

Problem

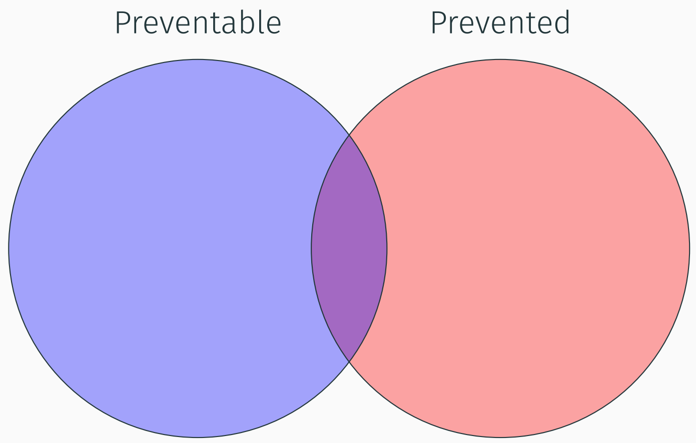
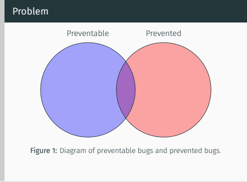


Figure 1: Diagram of preventable bugs and prevented bugs.


Problem

Preventable Prevented

Figure 1: Diagram of preventable bugs and prevented bugs.

Problem - with Rust

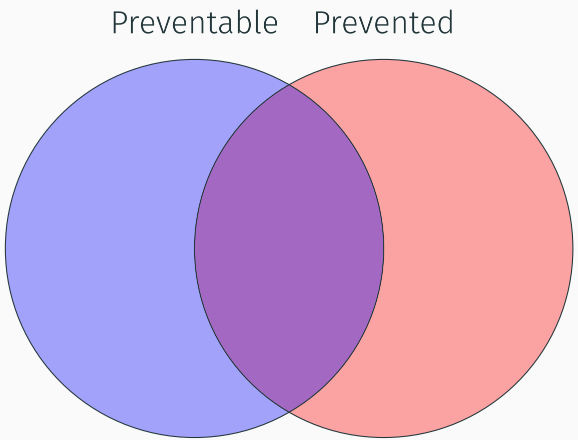


Figure 2: Diagram of preventable bugs and prevented bugs when considering Rust's borrow checker.

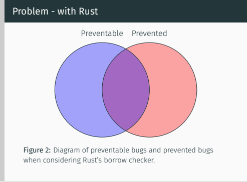

Problem - with Rust

Figure 2: Diagram of preventable bugs and prevented bugs when considering Rust's borrow checker.

Problem - Ideal

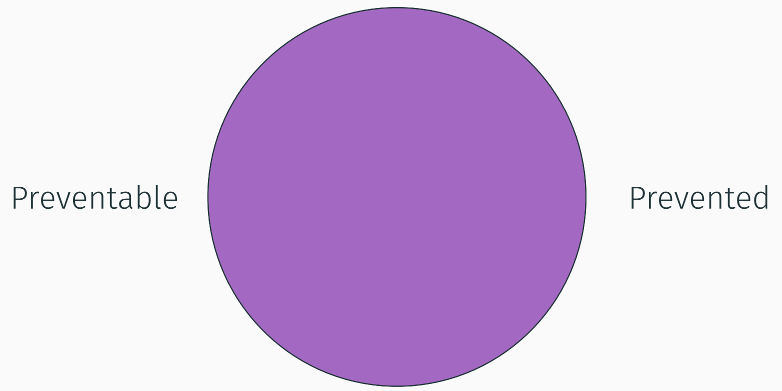


Figure 3: The ideal diagram of preventable bugs and prevented bugs, where all bugs are prevented.

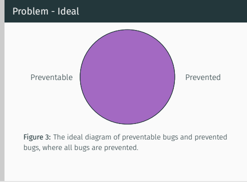
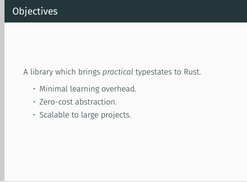

Problem - Ideal

Figure 3: The ideal diagram of preventable bugs and prevented bugs, where all bugs are prevented.

Objectives

A library which brings *practical* typestates to Rust.

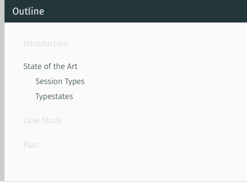
- Minimal learning overhead.
- Zero-cost abstraction.
- Scalable to large projects.


Objectives

A library which brings *practical* typestates to Rust.

- Minimal learning overhead.
- Zero-cost abstraction.
- Scalable to large projects.

Outline



Outline

Introduction

State of the Art

Session Types

Typestates

Case Study

Plan

Session Types

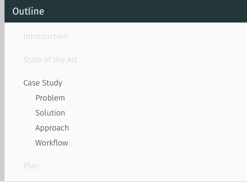
Bridging the gap between Typestates and Rust in production code
└─State of the Art
  └─Session Types

Typestates

2021-03-01

Bridging the gap between Typestates and Rust in production code
└─State of the Art
  └─Typestates

Outline


Outline

Introduction

State of the Art

Case Study

Problem

Solution

Approach

Workflow

Plan

└─Outline

Problem

Error happens at runtime, possibly crashing the program.

```rust
fn main() {
    let protocol = Protocol::new();
    protocol.step1();
    protocol.step3(); // runtime error
    protocol.step2();
}
```

Problem

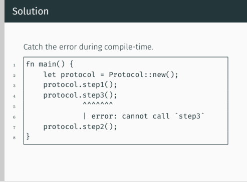
Error happens at runtime, possibly crashing the program.

```rust
fn main() {
    let protocol = Protocol::new();
    protocol.step1();
    protocol.step3(); // runtime error
    protocol.step2();
}
```

Solution

Catch the error during compile-time.

```
fn main() {
    let protocol = Protocol::new();
    protocol.step1();
    protocol.step3();
            ^^^^^^^
            | error: cannot call `step3`
    protocol.step2();
}
```


Solution

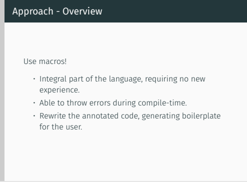
Catch the error during compile-time.

```
fn main() {
    let protocol = Protocol::new();
    protocol.step1();
    protocol.step3();
            ^^^^^^^
            | error: cannot call `step3`
    protocol.step2();
}
```

Approach - Overview

Use macros!

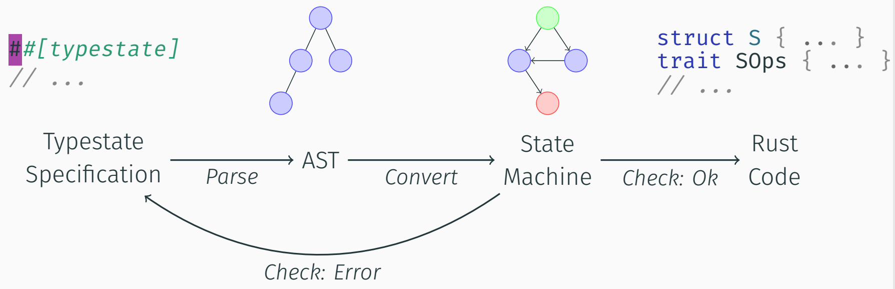
- Integral part of the language, requiring no new experience.
- Able to throw errors during compile-time.
- Rewrite the annotated code, generating boilerplate for the user.


Approach - Overview

Use macros!

- Integral part of the language, requiring no new experience.
- Able to throw errors during compile-time.
- Rewrite the annotated code, generating boilerplate for the user.

Approach - Going deeper



```
##[typestate]
// ...
```

```
struct S { ... }
trait SOps { ... }
// ...
```

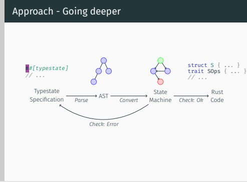
$$\begin{tikzcd}
\text{Typestate Specification} \arrow[r, "Parse"'] & \text{AST} \arrow[r, "Convert"'] & \text{State Machine} \arrow[r, "Check: Ok"'] \arrow[ll, bend left, "Check: Error"] & \text{Rust Code}
\end{tikzcd}$$

2021-03-01

Bridging the gap between Typestates and Rust in production code

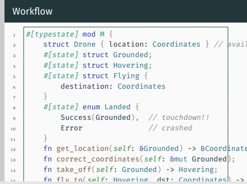
└Case Study

└Approach



Workflow

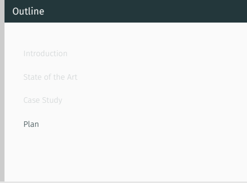
```
#[typestate] mod M {
    struct Drone { location: Coordinates } // avail
    #[state] struct Grounded;
    #[state] struct Hovering;
    #[state] struct Flying {
        destination: Coordinates
    }
    #[state] enum Landed {
        Success(Grounded),  // touchdown!!
        Error               // crashed
    }
    fn get_location(self: &Grounded) -> &Coordinate
    fn correct_coordinates(self: &mut Grounded);
    fn take_off(self: Grounded) -> Hovering;
    fn fly_to(self: Hovering, dst: Coordinates) ->
```


Workflow

```
#[typestate] mod M {
    struct Drone { location: Coordinates } // avail
    #[state] struct Grounded;
    #[state] struct Hovering;
    #[state] struct Flying {
        destination: Coordinates
    }
    #[state] enum Landed {
        Success(Grounded),  // touchdown!!
        Error               // crashed
    }
    fn get_location(self: &Grounded) -> &Coordinate
    fn correct_coordinates(self: &mut Grounded);
    fn take_off(self: Grounded) -> Hovering;
    fn fly_to(self: Hovering, dst: Coordinates) ->
```

Outline

2021-03-01


Outline

Introduction

State of the Art

Case Study

Plan

Plan Overview

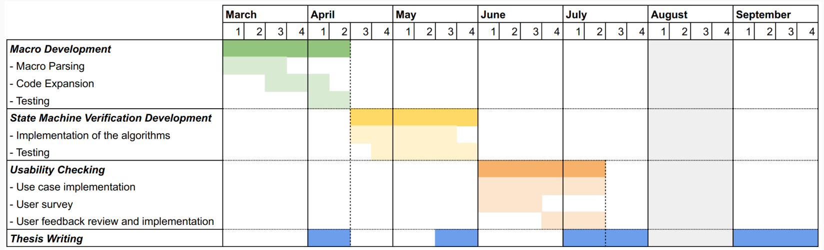
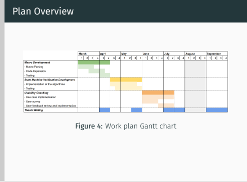


Figure 4: Work plan Gantt chart

2021-03-01

Bridging the gap between Typestates and Rust in production code

└Plan



Plan Overview

Figure 4: Work plan Gantt chart