Bridging the gap between Typestates and Rust in production code

José Duarte

António Ravara (Advisor)

March 2021

NOVA School of Science and Technology

2021-03-02

Bridging the gap between Typestates and Rust in production code

Bridging the gap between Typestates and Rust in production code

isé Duarte ntónio Ravara (Advisor) arch 2021 VA 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

2021

 $_{\mbox{\scriptsize B}}$ Bridging the gap between Typestates and Rust ㅎ in production code

Introduction

Outline

└─Outline

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.

- Introduction

 - State of the Art
 - Case Study
- Plan



Introduction

 $_{\mbox{\scriptsize 8}}$ Bridging the gap between Typestates and Rust

g in production code

—Introduction

A

2021-(

3

Outline

Context

Problem Objectives

Context

Problem Objectives

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.

Bridging the gap between Typestates and Rust in production code
Introduction
Context

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.

not nice.

A bug in the firmware for a pacemaker may cost a life.

Problem

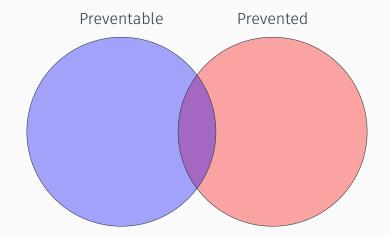


Figure 1: Diagram of preventable bugs and prevented bugs.

Bridging the gap between Typestates and Rust in production code

Introduction

Problem

D. . . l. l



Problem - with Rust

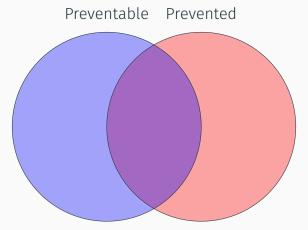


Figure 2: Diagram of preventable bugs and prevented bugs when considering Rust's borrow checker.

Bridging the gap between Typestates and Rust in production code
Introduction
Problem



Problem - Ideal

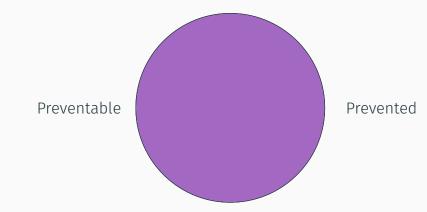


Figure 3: The ideal diagram of preventable bugs and prevented bugs, where all bugs are prevented.

Bridging the gap between Typestates and Rust in production code

Introduction

Problem



Objectives

A library which brings *practical* typestates to Rust.

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

Bridging the gap between Typestates and Rust in production code
Introduction
Objectives

Objectives

A library which brings practical typestates to Rust.

Zero-cost abstraction.

Scalable to large projects.

Outline

A

2021-(

 $_{\mbox{\scriptsize 8}}$ Bridging the gap between Typestates and Rust in production code State of the Art

Outline State of the Art Typestates

Session Types

State of the Art

Session Types

Typestates

Session Types

Bridging the gap between Typestates and Rust in production code

State of the Art

Session Types

Session Types

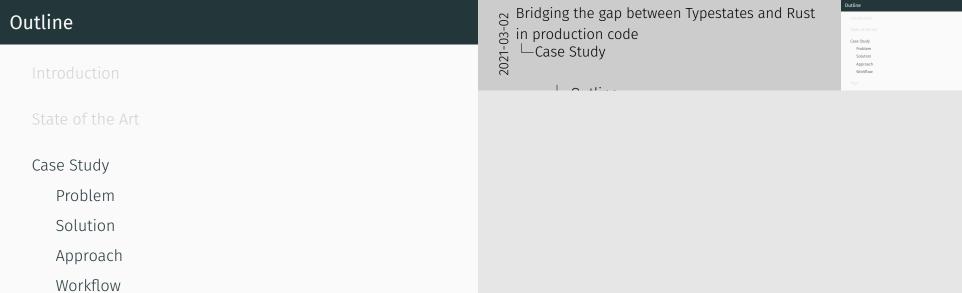
Typestates

Bridging the gap between Typestates and Rust in production code

State of the Art

Typestates

Typestates



Problem

Error happens at runtime, possibly crashing the program.

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

Our tools should work for us, not make us work for them.

Bridging the gap between Typestates and Rust in production code

Case Study

Problem

Error happens at runtime, possibly crashing the program.

fin main() {
 let protocol * Protocol:new();
 protocol.step1();
 protocol.step2();
 rotocol.step2();
}

Solution

Ideally, we want to catch the error at compile-time.

Bridging the gap between Typestates and Rust
in production code
Case Study
Solution

Solution

ideally, we want to catch the error at compile-time.

If n main() {
 let protocol = Protocol::new();
 protocol.steps();
 protocol.steps();
 let protocol.steps();
 protocol.steps();
 protocol.steps();
}

Approach - Overview

We can exploit the Rust typesystem to emulate typestates, however this approach requires boilerplate.

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.

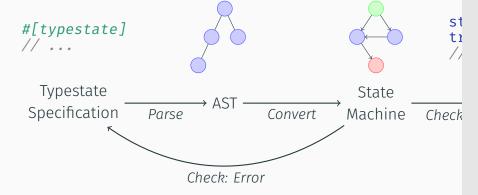
Bridging the gap between Typestates and Rust in production code -Case Study 2021--Approach

Approach - Overview

We can exploit the Rust typesystem to emulate typestates, however this approach requires boilerplate

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

Approach - Going deeper



Bridging the gap between Typestates and Rust in production code

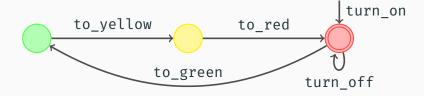
Case Study

Approach



Workflow - Design the state machine

Consider a traffic light as a state machine.



Bridging the gap between Typestates and Rust in production code

Case Study

Workflow



Workflow - Declaring state in Rust

Using the DSL we first declare the module, main automata and the states:

```
#[typestate] mod traffic_light {
    #[automata] struct TrafficLight;
    #[state] struct Green;
#[state] struct Yellow;
#[state] struct Red;
// ...
```

We still need transitions!

Bridging the gap between Typestates and Rust in production code

Case Study

Workflow

Workflow - Declaring state in Rust

Using the DSL we first declare the module, main auditorial and the states:

Elippostate	Dock	The First	Dock	Dock
Elippostate	Dock	Dock	Dock	
Elippostate	Dock	Dock	Dock	
Elippostate	Dock	Dock		
Elippostate	Dock	Dock		
Elippostate	Dock			
E				

Workflow - Declaring transitions in Rust

All transition functions take ownership of the current state and return the new state.

```
// code from the previous slide ...
fn to_yellow(self: Green) -> Yellow;
fn to_red(self: Yellow) -> Red;
fn to_green(self: Red) -> Green;
// ...
```

Finally, we need start and end states.

Bridging the gap between Typestates and Rust
in production code
Case Study
Workflow

Finally, we need start and end states.

Workflow - Declaring start and end states in Rust

Functions that do not use **self** and *return* a valid state are inferred as the *start* state.

Functions that take **self** and do *not return* a valid state are inferred as the *end* state.

```
// code from the previous slides ...
fn turn_on() -> Red;
fn turn_off(self: Red);
}
```

Our traffic light is ready!

Bridging the gap between Typestates and Rust in production code
Case Study
Workflow

Workflow - Declaring stort and end states in Rust
Functions that do not use self and return a valid state
are inferred as test state.
Functions that take self and do not return a valid state
are inferred as the end state.

// code from the previous slides ...
f nturn_enf() > Red;
fn turn_eff(self: Red);
}

traffic light is ready!

There are still other important features left. For maintenance purposed, consider that our traffic light is now required to count every **GYR** cycle.

The **TrafficLight** structure is now declared as follows:

#[automata] struct TrafficLight { cycles:u64 }

Bridging the gap between Typestates and Rust in production code
Case Study
Workflow

The sale of Co. 1

Workflow - The other features

There are still other important features left. For maintenance purposed, consider that our traffic light now required to count every GYR cycle.

The TrafficLight structure is now declared as

[automata] struct TrafficLight { cycles:u64

How can we check if the light requires maintenance?

We add a **pure** function:

This function is not able to perform mutations due to the immutable reference (&).

Bridging the gap between Typestates and Rust in production code
Case Study
Workflow

Workflow - The other features

How can we check if the light requires maintenance: We add a pure function:

function is not able to perform mutations due to the utable reference (Θ).

How can we check if the light requires maintenance?

We add a **pure** function:

```
fn requires_maintenance(
    self: &TrafficLight
) -> bool;
```

This function is not able to perform mutations due to the immutable reference (&).

Bridging the gap between Typestates and Rust in production code

Case Study

Workflow

Workflow - The other features

How can we check if the light requires maintenance: We add a pure function:

fn requires_maintenance(
 self: &TrafficLight
) -> bool;

This function is not able to perform mutations due to the mmutable reference (a).

After maintenance, how can we reset the counter?
We add an **impure** function:

fn reset_counter(self: &mut TrafficLight);

The function is able to perform mutations, due to usage of a mutable reference (**&mut**), but it is unable to transition between states.

Bridging the gap between Typestates and Rust in production code

Case Study

Workflow

Workflow - The other features

We add an impure function:

fn reset_counter(self: &mut TrafficLight)

The function is able to perform mutations, due to usage of a mutable reference (&mut), but it is unable to transition between states.



Bridging the gap between Typestates and Rust in production code
Plan

Outline

Plan

- Plan

25

- L 0.41:...

Plan Overview

					_																				_			_
	Mar	rch			April				May				June				July				August				September			
	1	2	2 3	4	- 1	2	3	4	1	2	3	4	- 1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	3
Macro Development													Г															
- Macro Parsing																												
- Code Expansion																												
- Testing																												
State Machine Verification Development									Ī																Ī			
- Implementation of the algorithms																												
- Testing																												
Usability Checking	T								T																Ī			
- Use case implementation																												
- User survey																												
- User feedback review and implementation																												
Thesis Writing																												

Figure 4: Work plan Gantt chart

Bridging the gap between Typestates and Rust in production code
Plan

DI--- 0-----

Padil Overview

| Page | Page