MUTATIS

Mutation Testing Development Tool

(for **Anchor** programs)

Franck MAUSSAND: franck@maussand.net

GitHub: https://github.com/Laugharne/mutatis_mutandis

”Mutatis Mutandis”

Mutatis Mutandis is a Medieval Latin phrase meaning:

”with things changed that should be changed”

or

”once the necessary changes have been made”

Source: https://en.wikipedia.org/wiki/Mutatis_mutandis

TABLE OF CONTENTS

DEMO DAY:

”Solana Summer Fellowship 2024”

The fellowship ends with a Demo Day, which brings an opportunity to receive grants, funding, or otherwise work with the hottest projects in the Solana ecosystem.

Who am i ?

- Industrial computing (low level coding, noises & vibrations analysis)
- PHP Backend (E-Commerce)
- 2D/3D interactive development (game around Y2K)
- Blockchain learning: **Alyra School** x **Solana Foundation** (EVM & Solana) / **Rust** improvement
- **Technical posts** about **blockchain** published on **Medium** also contributor for **CoinsBench !**

What is Mutation Testing ?

It’s a technique used to **evaluate and improve** the quality of your **unit tests**.

It involves introducing small changes (***mutations***) to your code, and then checking if your tests can detect these changes by failing.

If a test passes despite a mutation, it indicates that the test suite might not be comprehensive enough.

What is Mutation Testing ?

It provides an **additional layer of confidence in the codebase** and helps deliver high-quality software that meets the expectations of users and stakeholders.

Although mutation testing is time-consuming, integrating it with an automation tools can enhance the effectiveness of the testing process and contribute to the development of more resilient and trustworthy software systems.

How it works ?

1. Original code
2. Introduce mutation
3. Run tests
4. Check results

How it works ?

Original code:

You have a function or a set of functions covered by unit tests.

Here’s a simple code example in Rust:

```rust
fn is_even(x: u32) -> bool {
    x % 2 == 0
}
```

How it works ?

Introduce mutation

Small changes are made to the code (*e.g., flipping a boolean, changing an operator*)

A **mutation** could be flipping the **boolean condition**:

```
fn is_even(x: u32) → bool {
    x % 2 ≠ 0  // Changed from `==` to `≠`
}
```

How it works ?

Run tests and check results

- If the **tests fail**, they "*kill*" the mutation, meaning the **test suite is effective**.
- If the **tests pass**, the mutation "*survives*," indicating that the **test suite** might **not cover** that part of **the code well**.

How it works ?

Benefits:

- Helps ensure that your tests are meaningful.
- Encourages stronger, more effective tests.

Mutation testing tools automate this process by applying mutations and running your test suite.

Mutatis

Mutatis is a **CLI development tool**, build in **Rust** for Rust **Anchor** program.

It requires a fully functional **Anchor program** and its **associated tests**.

The program must imperatively pass its own test units successfully.

The goal is precisely to **disrupt the test units** in order to be able to **detect flaws**!

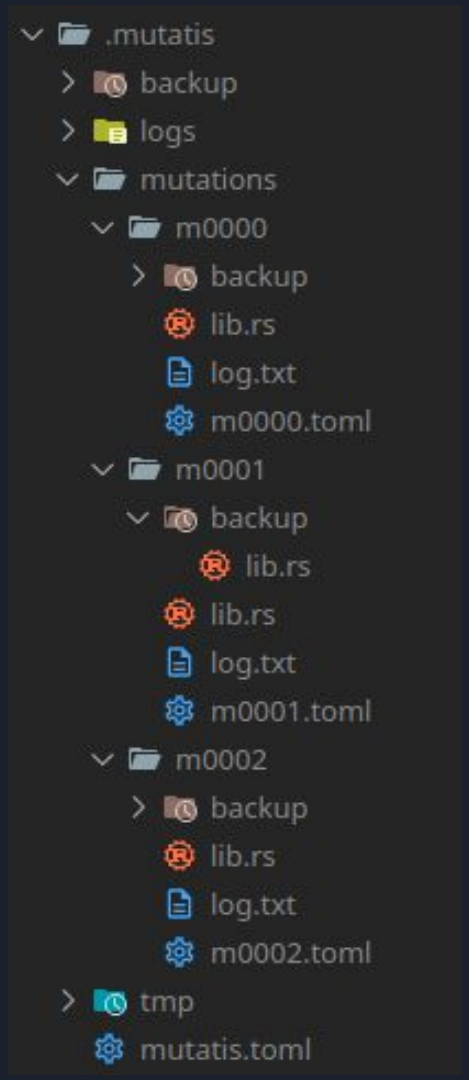
Mutatis

Three main commands:

- `mutatis init` : initialize & prepare a project for mutations, you can set some specific parameters
- `mutatis analyze` : proceed to source code analyze and generate code mutations
- `mutatis run` : performs source code testing with mutated source codes and produces feedback

Mutatis

A new arboresence is created, dedicated to mutations



Mutatis

Initial source code

(*before mutations*)

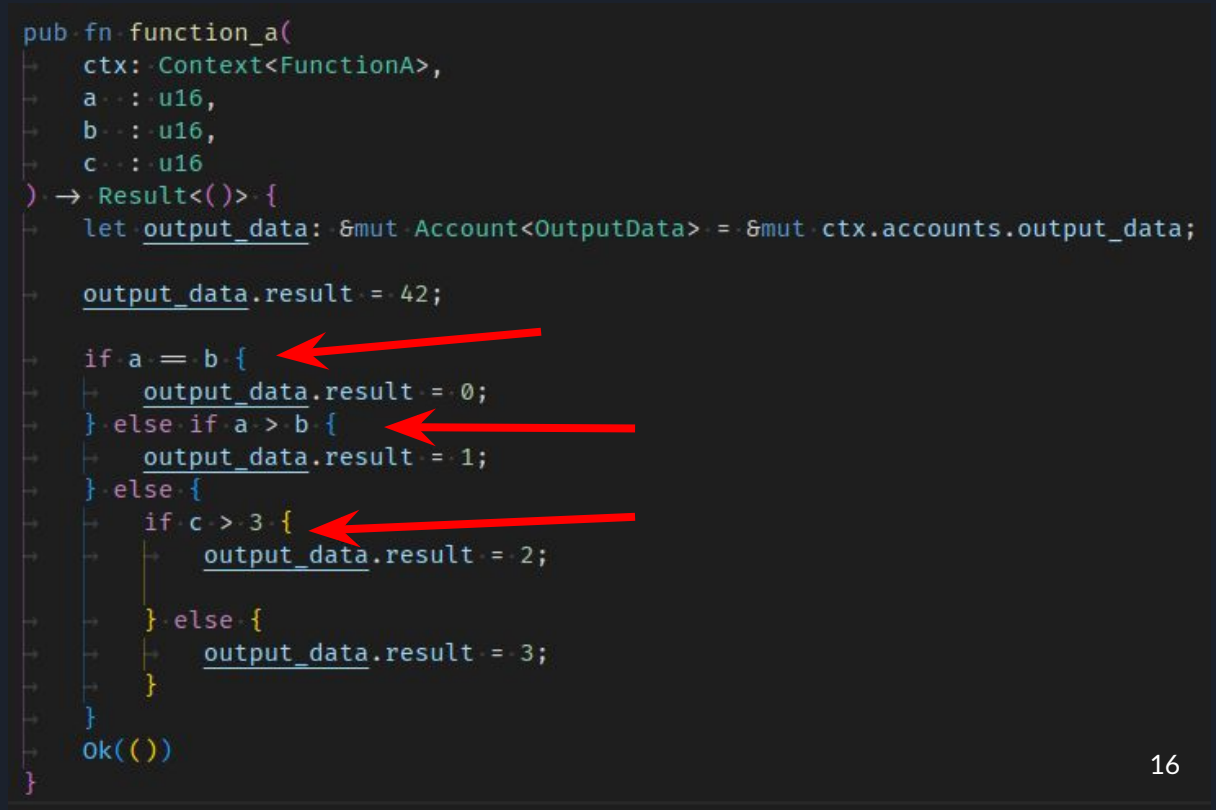
```rust
pub fn function_a(
    ctx: Context<FunctionA>,
    a  : u16,
    b  : u16,
    c  : u16
) -> Result<()> {
    let output_data: &mut Account<OutputData> = &mut ctx.accounts.output_data;

    output_data.result = 42;

    if a == b {
        output_data.result = 0;
    } else if a > b {
        output_data.result = 1;
    } else {
        if c > 3 {
            output_data.result = 2;

        } else {
            output_data.result = 3;
        }
    }
    Ok(())
}
```

Mutatis

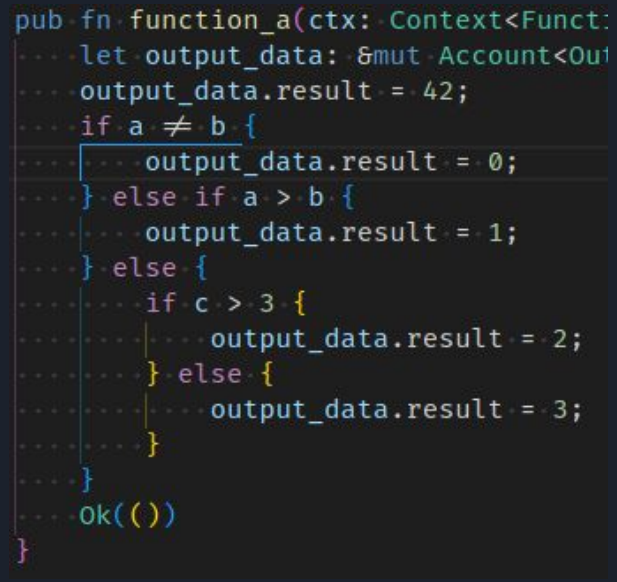
Initial source code

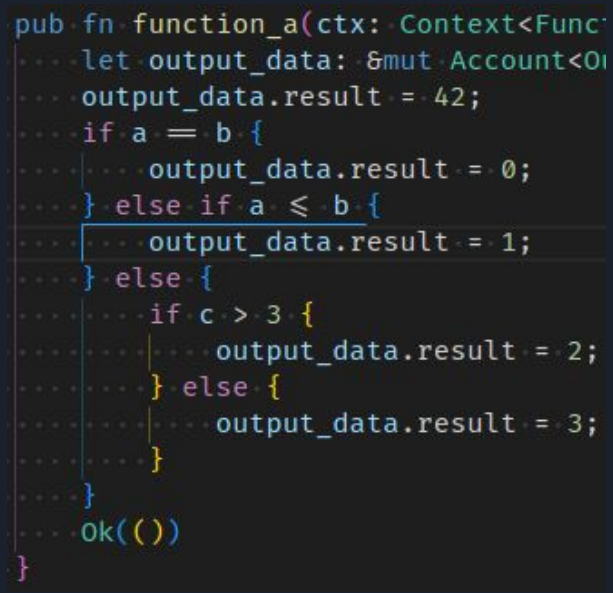
Three entry points

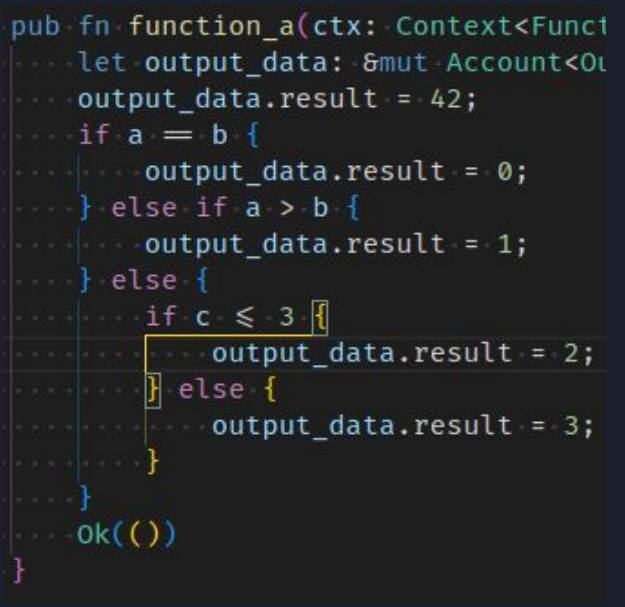
for mutations



Mutatis

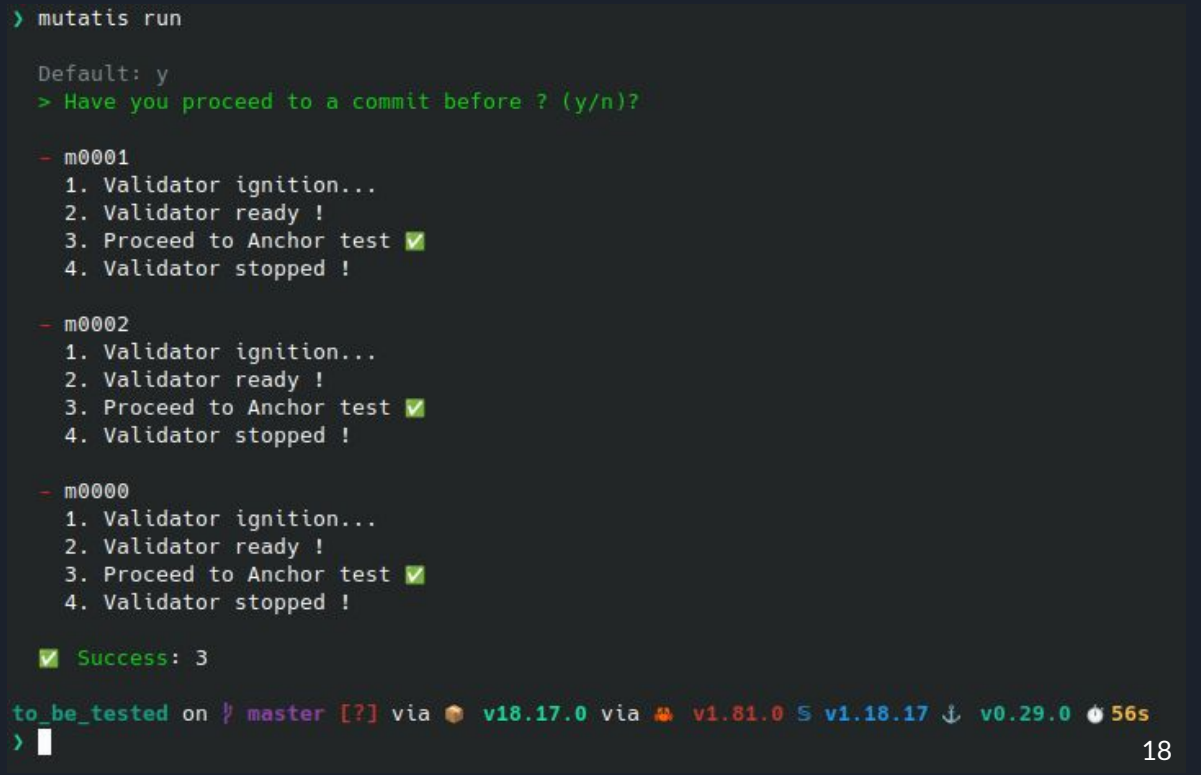

pub fn function_a(ctx: Context<Funct
let output_data: &mut Account<Ou
output_data.result = 42;
if a ≠ b {
output_data.result = 0;
} else if a > b {
output_data.result = 1;
} else {
if c > 3 {
output_data.result = 2;
} else {
output_data.result = 3;
}
}
Ok(())
}


pub fn function_a(ctx: Context<Funct
let output_data: &mut Account<Ou
output_data.result = 42;
if a == b {
output_data.result = 0;
} else if a ⩽ b {
output_data.result = 1;
} else {
if c > 3 {
output_data.result = 2;
} else {
output_data.result = 3;
}
}
Ok(())
}


pub fn function_a(ctx: Context<Funct
let output_data: &mut Account<Ou
output_data.result = 42;
if a == b {
output_data.result = 0;
} else if a > b {
output_data.result = 1;
} else {
if c ⩽ 3 {
output_data.result = 2;
} else {
output_data.result = 3;
}
}
Ok(())
}

Mutatis

Mutatis automate this process by applying mutations and running your test suite.

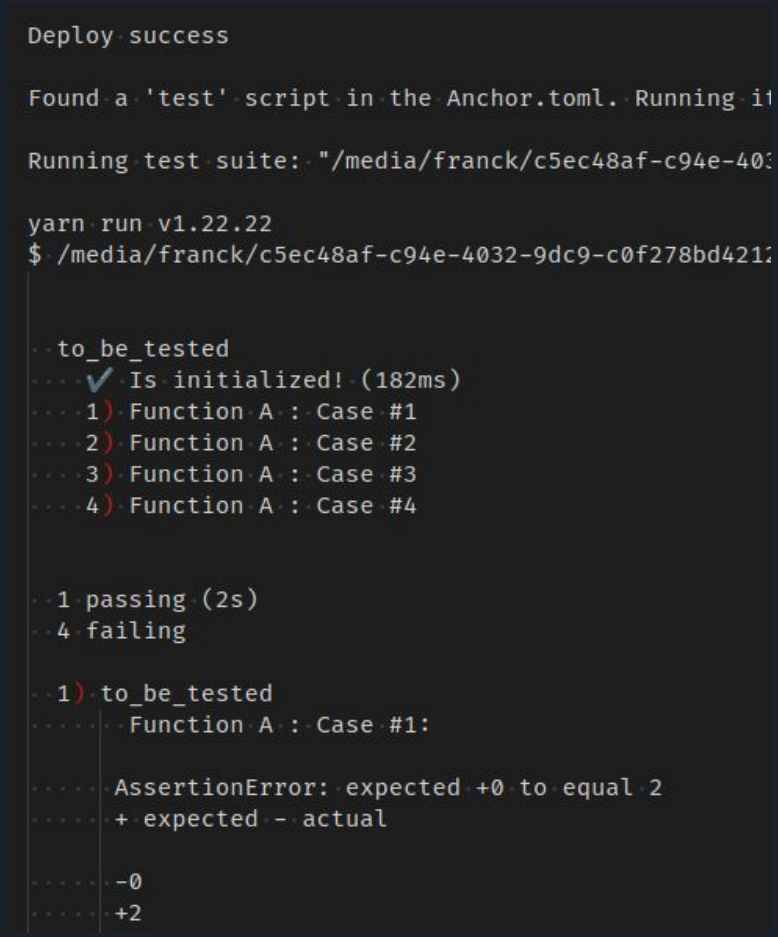


Mutatis

We can then,

get the logs from

anchor test



Technical Insight

- CLI management
- Rust/Cargo
- AST (Abstract Symbolic Tree) handling
- Writing/reading TOML files
- Background processes control
- Anchor tool control

Demonstration

To do

Using **Jest** and **Bankrun** instead of Anchor Test to **increase speed**

More mutations per code entry point

Extension to **Native Rust** program

Using **more recent crates** to perform the analysis

Increase robustness

Links

- Github.io: https://laugharne.github.io/
- LinkedIn: https://www.linkedin.com/in/franckmaussand/
- GitHub: https://github.com/Laugharne
- Medium: https://medium.com/@franck.maussand/
- Mutatis: https://github.com/Laugharne/mutatis_mutandis

Thank You !

Solana Summer Fellowship 2024