STATE MACHINES
✨ STATE MACHINES ✨

William Brown: Senior Software Engineer, SUSE Labs

tw: @Erstejahre
email: wbrown@suse.de
https://github.com/Firstyear/purplecon_state_machines

Programming is hard

tw: @Erstejahre
email: wbrown@suse.de
https://github.com/Firstyear/purplecon_state_machines

- Release Schedules
- Maintainability
- Social Awareness
- Business requirements
- User Experience
- Debugging
- Team Knowledge Sharing
- Documentation

- Supportability
- Upgrade/Downgrade path
- Backups/Restore/DR
- ABI/API stability
- Performance
- Terrible Language Limitations
- Third Party Libraries?
- Robustness (Will it crash?)
  - Security (Will it be hax?)

tw: @Erstejahre
email: wbrown@suse.de
https://github.com/Firstyear/purplecon_state_machines

robust adjective

ro·bust | \ rō-ˈbəst 🔊, ˈrō-(ˌ)bəst\

c : strongly formed or constructed : STURDY

// a robust plastic

d : capable of performing without failure under a wide range of conditions

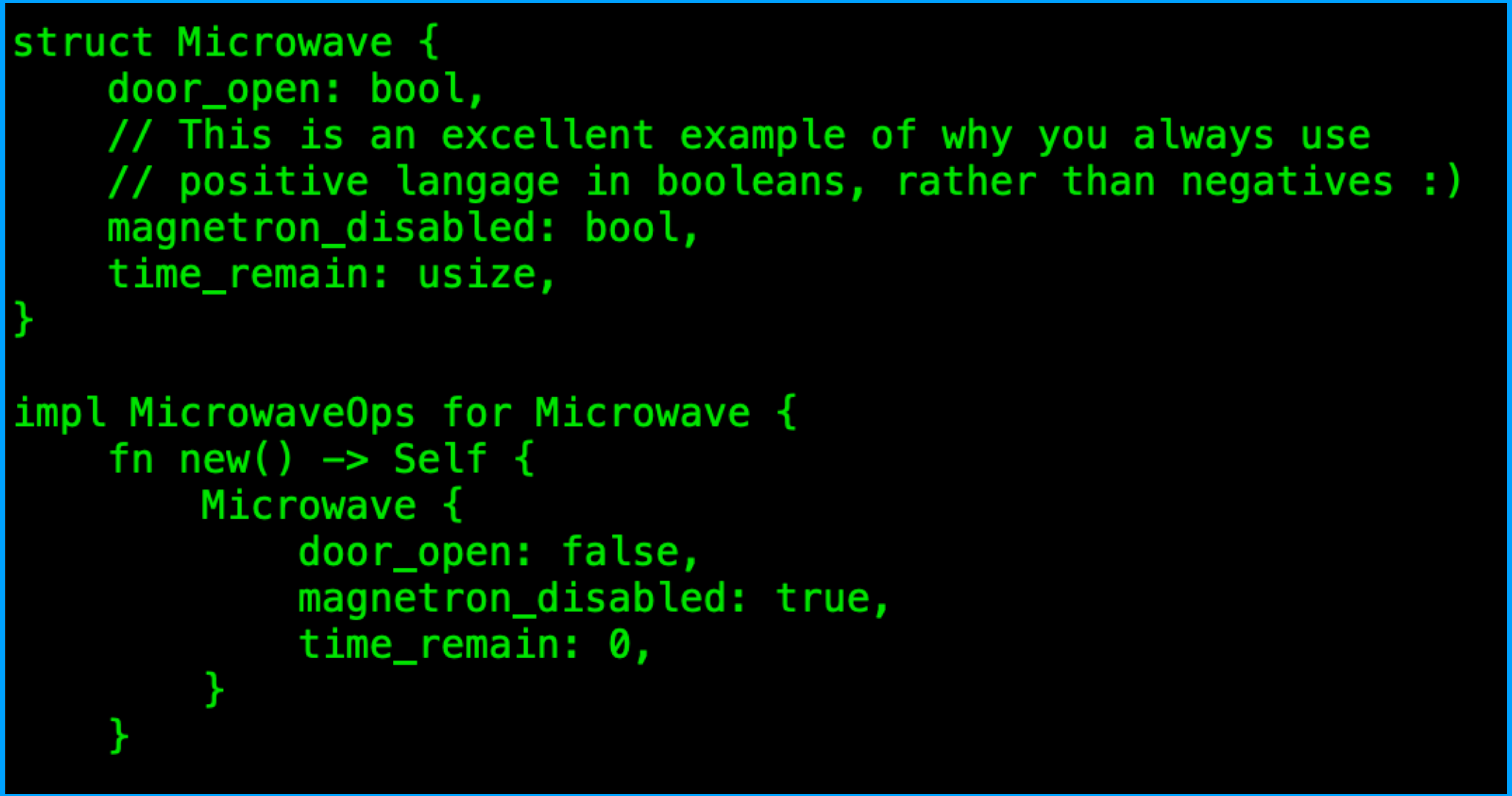
// robust software

ref: https://www.merriam-webster.com/dictionary/robust

tw: @Erstejahre
email: wbrown@suse.de
https://github.com/Firstyear/purplecon_state_machines

tw: @Erstejahre
email: wbrown@suse.de
https://github.com/Firstyear/purplecon_state_machines

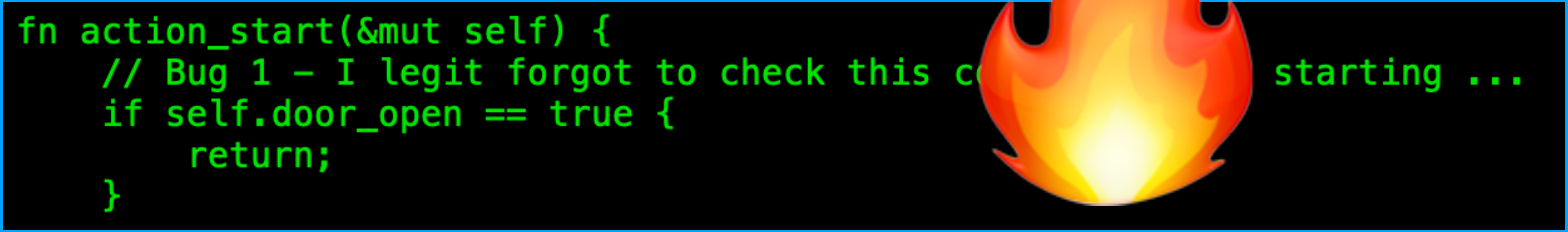
The Naivewave



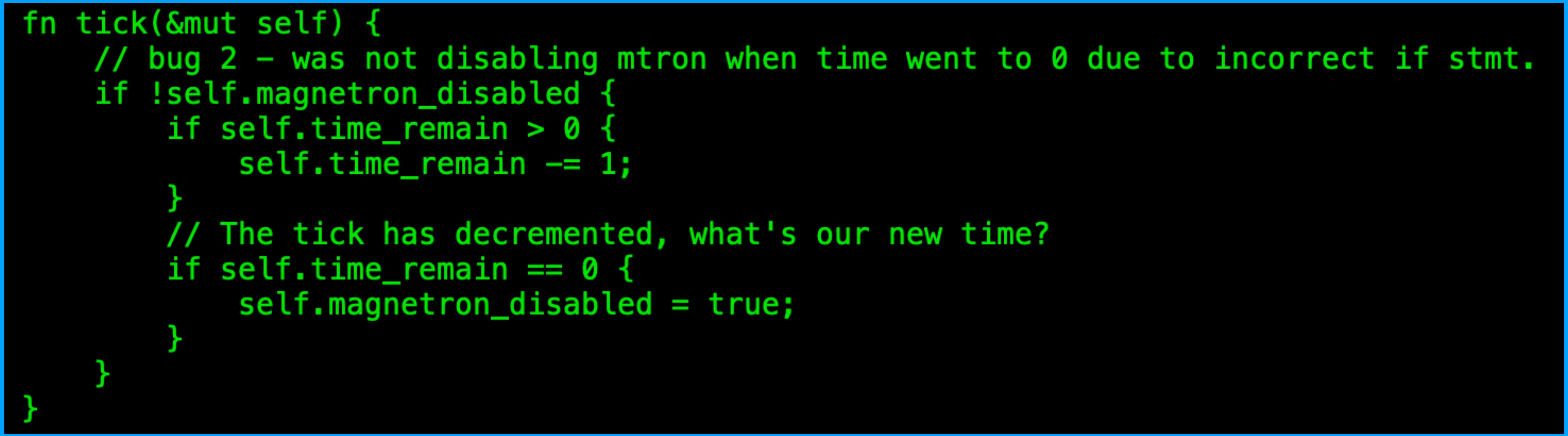
```
struct Microwave {
    door_open: bool,
    // This is an excellent example of why you always use
    // positive langage in booleans, rather than negatives :)
    magnetron_disabled: bool,
    time_remain: usize,
}

impl MicrowaveOps for Microwave {
    fn new() -> Self {
        Microwave {
            door_open: false,
            magnetron_disabled: true,
            time_remain: 0,
        }
    }
```

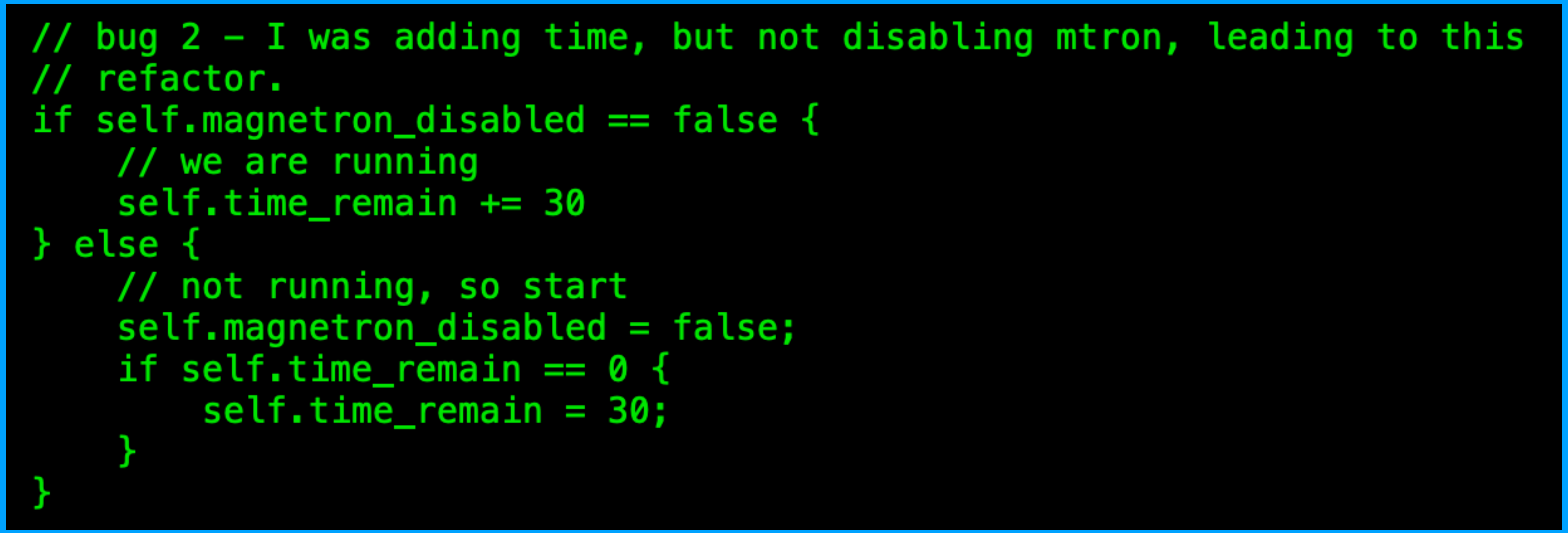
tw: @Erstejahre
email: wbrown@suse.de
https://github.com/Firstyear/purplecon_state_machines



```
fn action_start(&mut self) {
    // Bug 1 - I legit forgot to check this c            starting ...
    if self.door_open == true {
        return;
    }
```

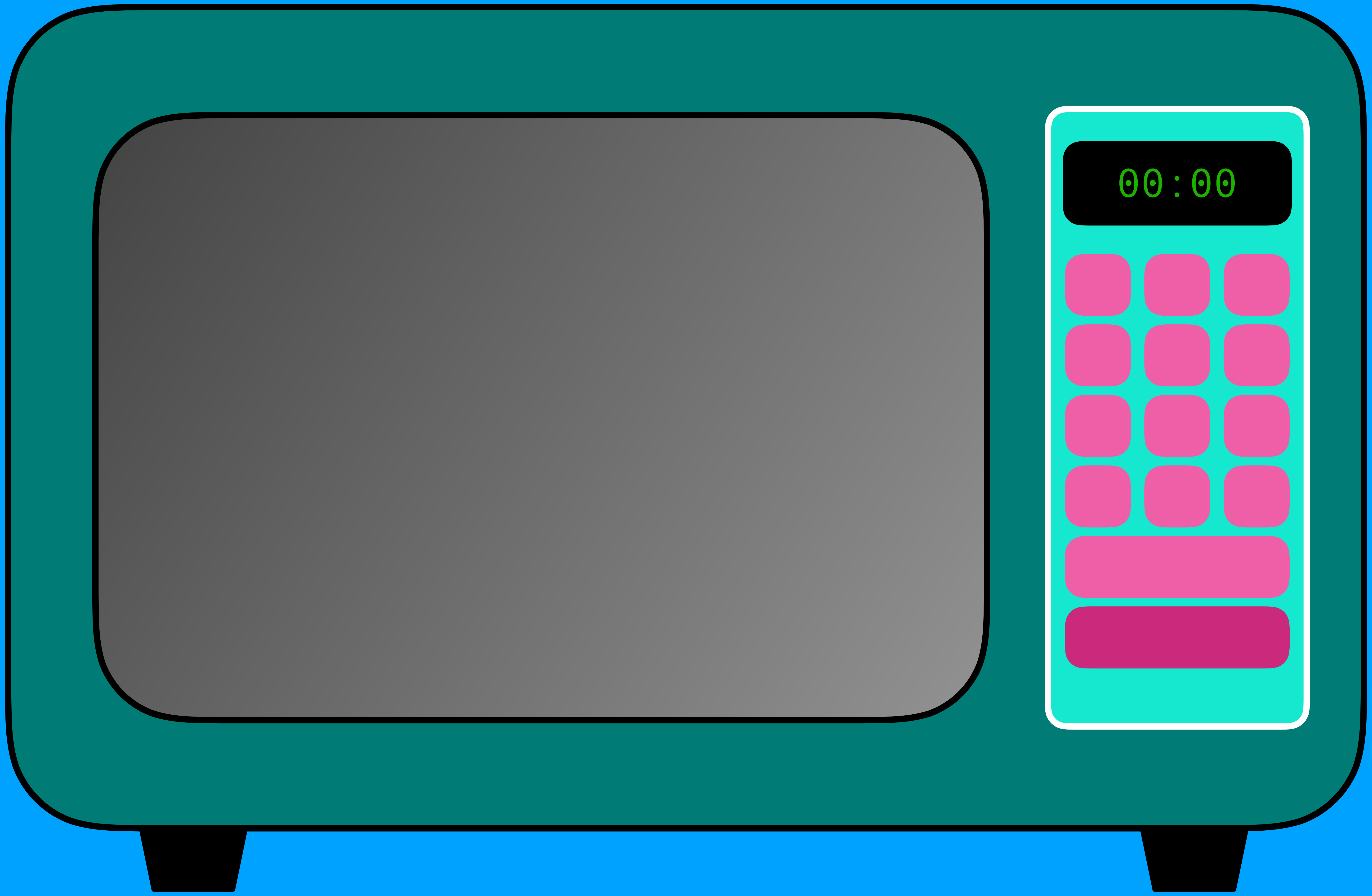


```
fn tick(&mut self) {
    // bug 2 - was not disabling mtron when time went to 0 due to incorrect if stmt.
    if !self.magnetron_disabled {
        if self.time_remain > 0 {
            self.time_remain -= 1;
        }
        // The tick has decremented, what's our new time?
        if self.time_remain == 0 {
            self.magnetron_disabled = true;
        }
    }
}
```



```
// bug 2 - I was adding time, but not disabling mtron, leading to this
// refactor.
if self.magnetron_disabled == false {
    // we are running
    self.time_remain += 30
} else {
    // not running, so start
    self.magnetron_disabled = false;
    if self.time_remain == 0 {
        self.time_remain = 30;
    }
}
```

tw: @Erstejahre
email: wbrown@suse.de
https://github.com/Firstyear/purplecon_state_machines



tw: @Erstejahre
email: wbrown@suse.de
https://github.com/Firstyear/purplecon_state_machines

The Stateful-wave

- Door Open - No Time Set
- Door Open - Time Set
- Door Close - No Time Set
- Door Close - Time Set
- Door Close - Time Set, Magnetron - cooking

The Stateful-wave

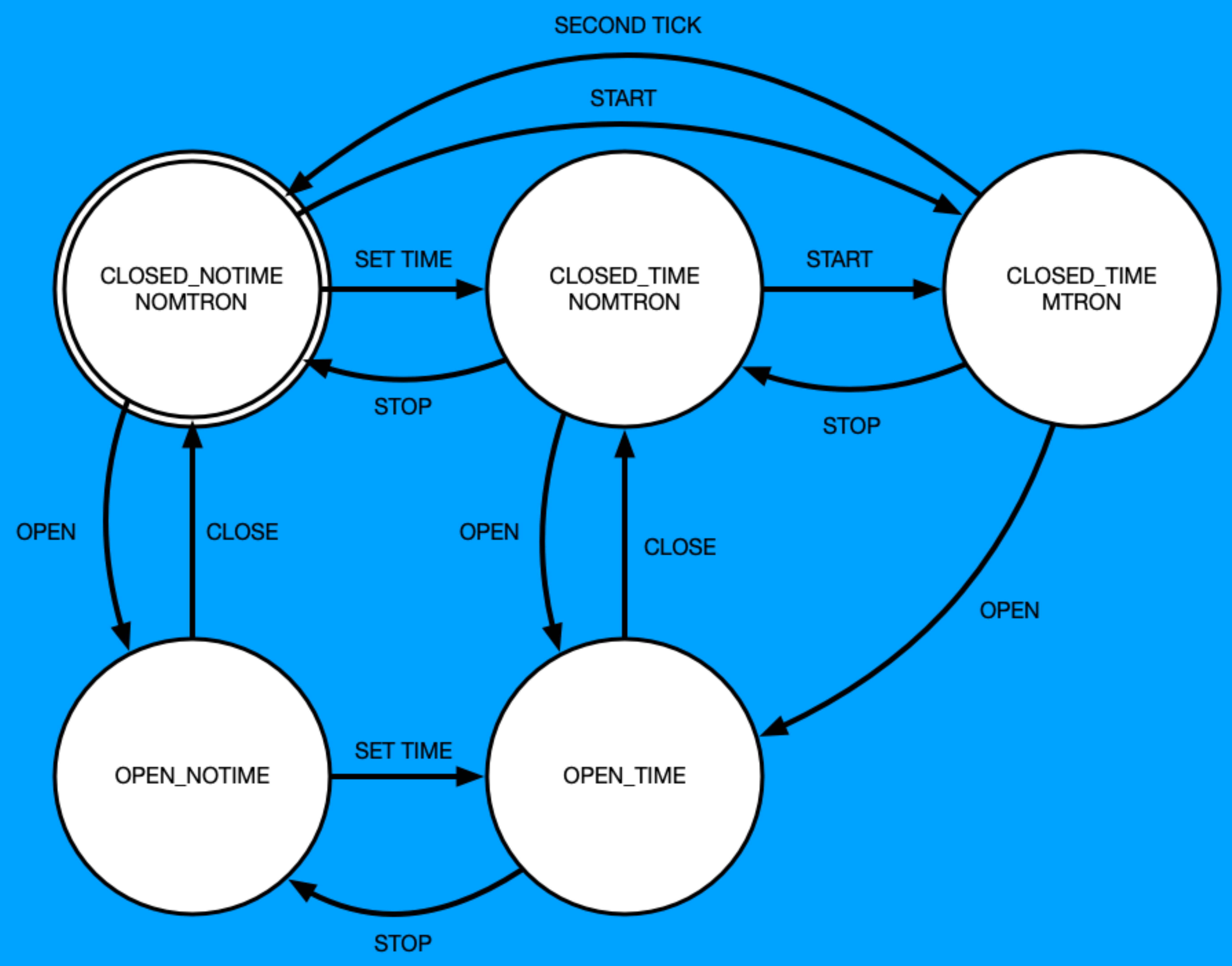
	OPEN_NOTIME	OPEN_TIME	CLOSED_NOTIME_NOMTRON	CLOSED_TIME_NOMTRON	CLOSED_TIME_MTRON
open door	OPEN_NOTIME	OPEN_TIME	OPEN_NOTIME	OPEN_TIME	OPEN_TIME
close door	CLOSED_NOTIME_NOMTRON	CLOSED_TIME_NOMTRON	CLOSED_NOTIME_NOMTRON	CLOSED_TIME_NOMTRON	CLOSED_TIME_MTRON
set time	OPEN_TIME	OPEN_TIME	CLOSED_TIME_NOMTRON	CLOSED_TIME_NOMTRON	CLOSED_TIME_MTRON
stop	OPEN_NOTIME	OPEN_NOTIME	CLOSED_NOTIME_NOMTRON	CLOSED_NOTIME_NOMTRON	CLOSED_TIME_NOMTRON
start	OPEN_NOTIME	OPEN_TIME	CLOSED_TIME_MTRON	CLOSED_TIME_MTRON	CLOSED_TIME_MTRON
one second elapses	OPEN_NOTIME	OPEN_TIME	CLOSED_NOTIME_NOMTRON	CLOSED_TIME_NOMTRON	CLOSED_TIME_MTRON CLOSED_NOTIME_MTRON

tw: @Erstejahre
email: wbrown@suse.de
https://github.com/Firstyear/purplecon_state_machines

The Stateful-wave

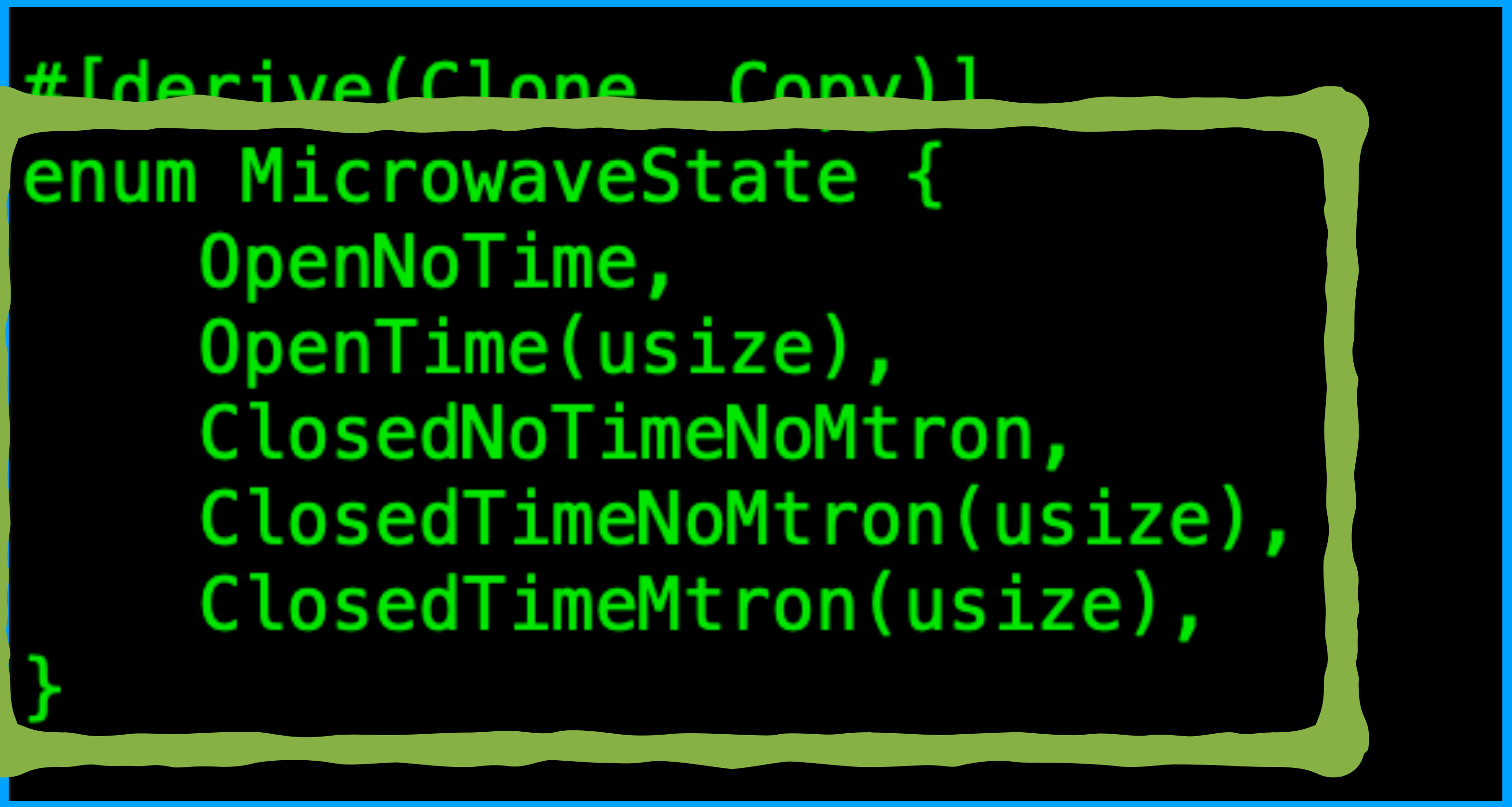
	OPEN_NOTIME	OPEN_TIME	CLOSED_NOTIME_NOMTRON	CLOSED_TIME_NOMTRON	CLOSED_TIME_MTRON
open door	-	-	OPEN_NOTIME	OPEN_TIME	OPEN_TIME
close door	CLOSED_NOTIME_NOMTRON	CLOSED_TIME_NOMTRON	-	-	-
set time	OPEN_TIME	-	CLOSED_TIME_NOMTRON	-	-
stop	-	OPEN_NOTIME	-	CLOSED_NOTIME_NOMTRON	CLOSED_TIME_NOMTRON
start	-	-	CLOSED_TIME_MTRON	CLOSED_TIME_MTRON	CLOSED_TIME_MTRON
one second elapses	-	-	-	-	CLOSED_TIME_MTRON CLOSED_NOTIME_MTRON

tw: @Erstejahre
email: wbrown@suse.de
https://github.com/Firstyear/purplecon_state_machines

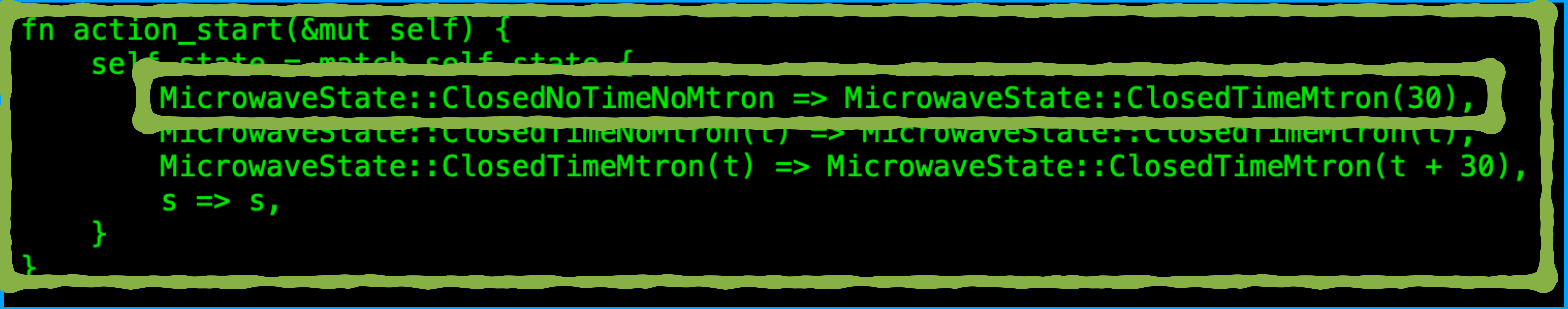


tw: @Erstejahre
email: wbrown@suse.de
https://github.com/Firstyear/purplecon_state_machines

Rust!



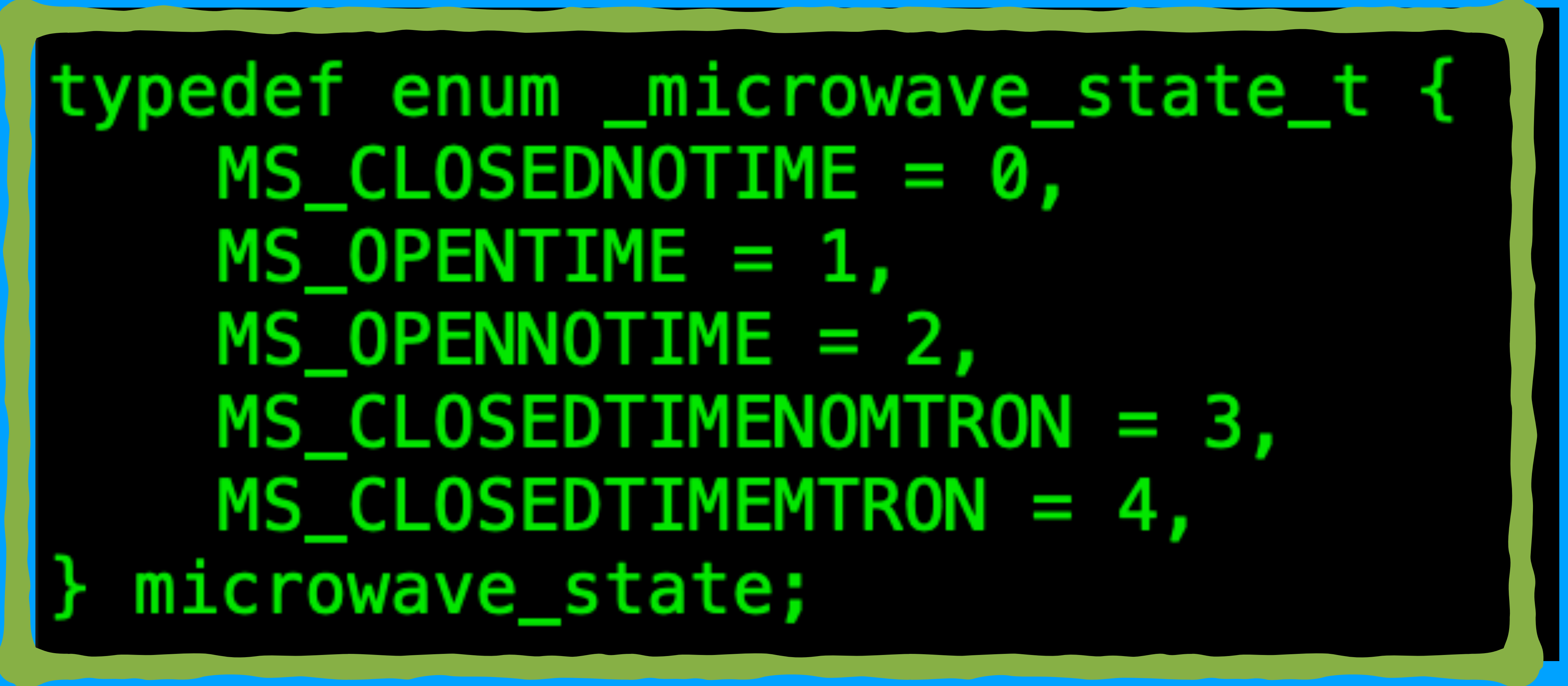
```
#[derive(Clone, Copy)]
enum MicrowaveState {
    OpenNoTime,
    OpenTime(usize),
    ClosedNoTimeNoMtron,
    ClosedTimeNoMtron(usize),
    ClosedTimeMtron(usize),
}
```



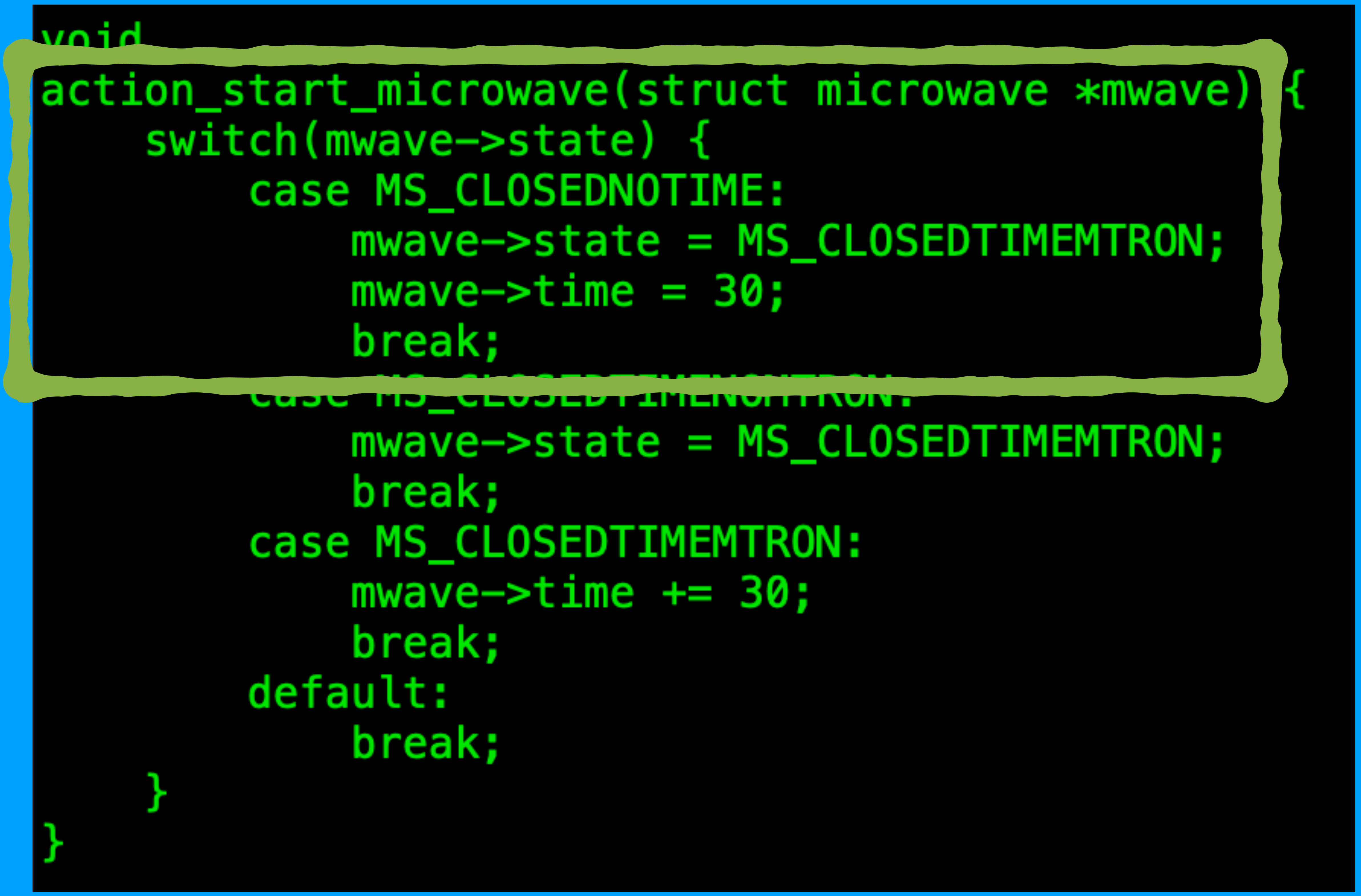
```
fn action_start(&mut self) {
    self.state = match self.state {
        MicrowaveState::ClosedNoTimeNoMtron => MicrowaveState::ClosedTimeMtron(30),
        MicrowaveState::ClosedTimeNoMtron(t) => MicrowaveState::ClosedTimeMtron(t),
        MicrowaveState::ClosedTimeMtron(t) => MicrowaveState::ClosedTimeMtron(t + 30),
        s => s,
    }
}
```

tw: @Erstejahre
email: wbrown@suse.de
https://github.com/Firstyear/purplecon_state_machines

Get into the C



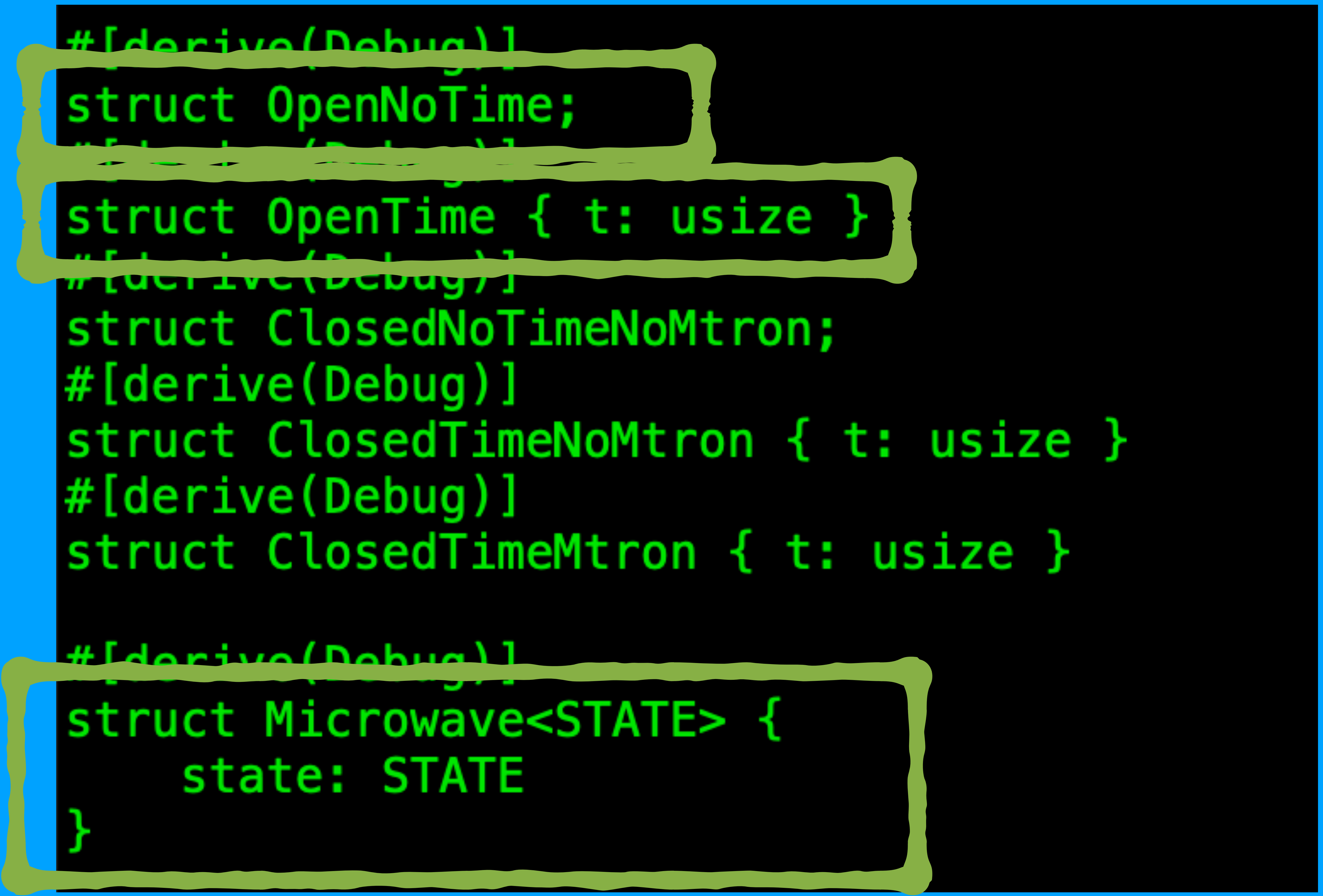
```
typedef enum _microwave_state_t {
    MS_CLOSEDNOTIME = 0,
    MS_OPENTIME = 1,
    MS_OPENNOTIME = 2,
    MS_CLOSEDTIMENOMTRON = 3,
    MS_CLOSEDTIMEMTRON = 4,
} microwave_state;
```



```
void
action_start_microwave(struct microwave *mwave) {
    switch(mwave->state) {
        case MS_CLOSEDNOTIME:
            mwave->state = MS_CLOSEDTIMEMTRON;
            mwave->time = 30;
            break;
        case MS_CLOSEDTIMENOMTRON:
            mwave->state = MS_CLOSEDTIMEMTRON;
            break;
        case MS_CLOSEDTIMEMTRON:
            mwave->time += 30;
            break;
        default:
            break;
    }
}
```

tw: @Erstejahre
email: wbrown@suse.de
https://github.com/Firstyear/purplecon_state_machines

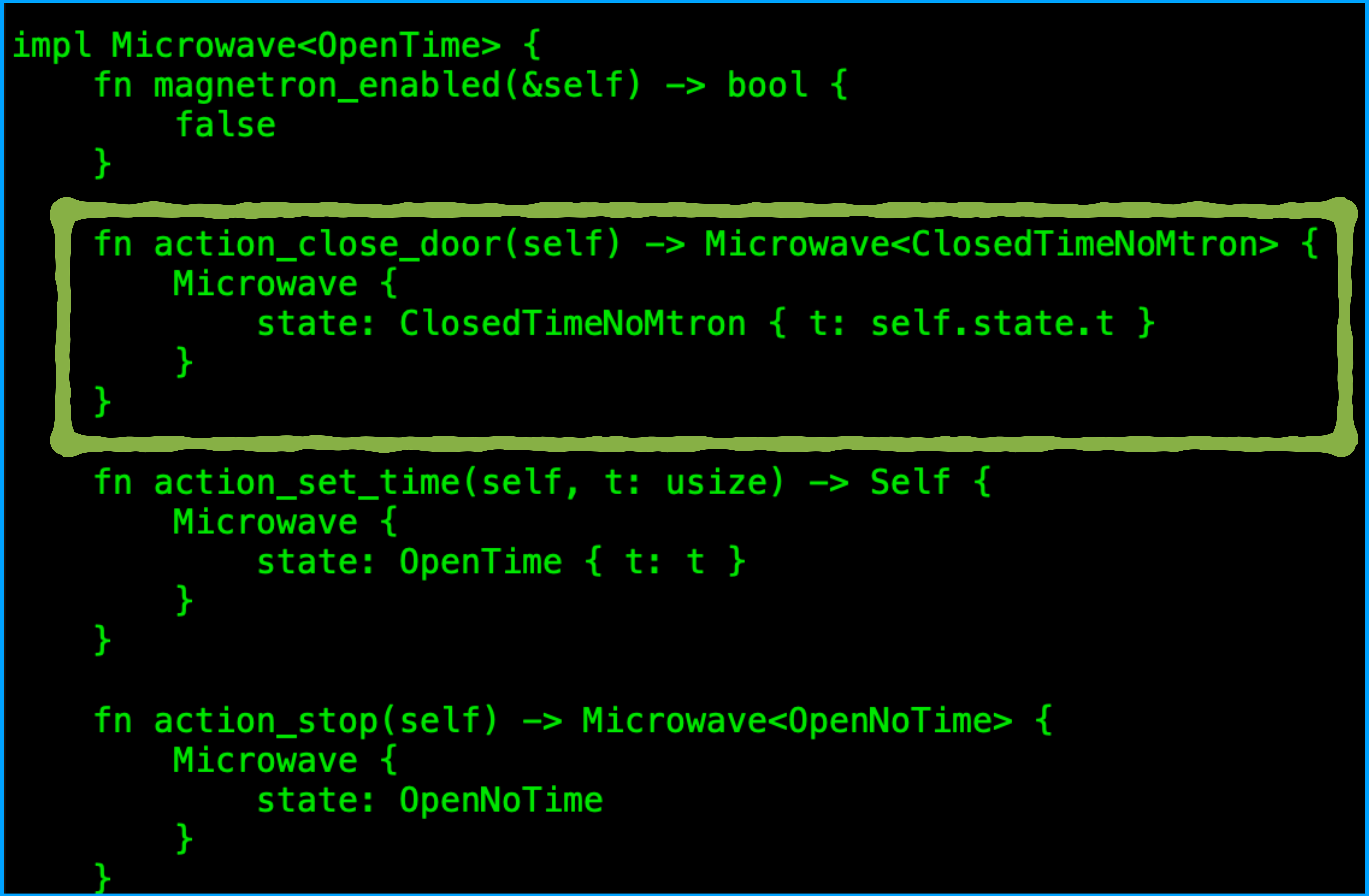
Other Approaches - Rust compiled



```
#[derive(Debug)]
struct OpenNoTime;
#[derive(Debug)]
struct OpenTime { t: usize }
#[derive(Debug)]
struct ClosedNoTimeNoMtron;
#[derive(Debug)]
struct ClosedTimeNoMtron { t: usize }
#[derive(Debug)]
struct ClosedTimeMtron { t: usize }

#[derive(Debug)]
struct Microwave<STATE> {
    state: STATE
}
```

tw: @Erstejahre
email: wbrown@suse.de
https://github.com/Firstyear/purplecon_state_machines



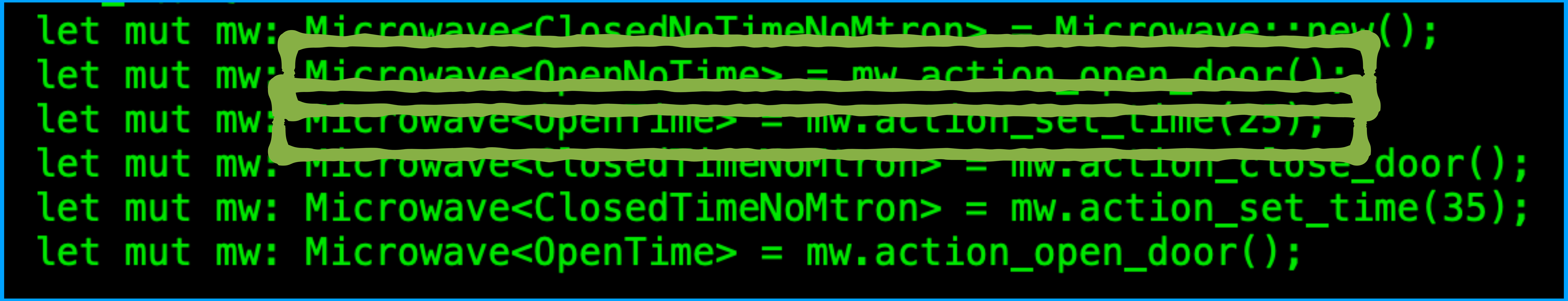
```
impl Microwave<OpenTime> {
    fn magnetron_enabled(&self) -> bool {
        false
    }

    fn action_close_door(self) -> Microwave<ClosedTimeNoMtron> {
        Microwave {
            state: ClosedTimeNoMtron { t: self.state.t }
        }
    }

    fn action_set_time(self, t: usize) -> Self {
        Microwave {
            state: OpenTime { t: t }
        }
    }

    fn action_stop(self) -> Microwave<OpenNoTime> {
        Microwave {
            state: OpenNoTime
        }
    }
```

tw: @Erstejahre
email: wbrown@suse.de
https://github.com/Firstyear/purplecon_state_machines



```
let mut mw: Microwave<ClosedNoTimeNoMtron> = Microwave::new();
let mut mw: Microwave<OpenNoTime> = mw.action_open_door();
let mut mw: Microwave<OpenTime> = mw.action_set_time(25);
let mut mw: Microwave<ClosedTimeNoMtron> = mw.action_close_door();
let mut mw: Microwave<ClosedTimeNoMtron> = mw.action_set_time(35);
let mut mw: Microwave<OpenTime> = mw.action_open_door();
```

tw: @Erstejahre
email: wbrown@suse.de
https://github.com/Firstyear/purplecon_state_machines

Option, Null, Bool, Int ... are states too!

- Some(T), None
- NULL, anything else
- true, false
- 0, 1, ∞

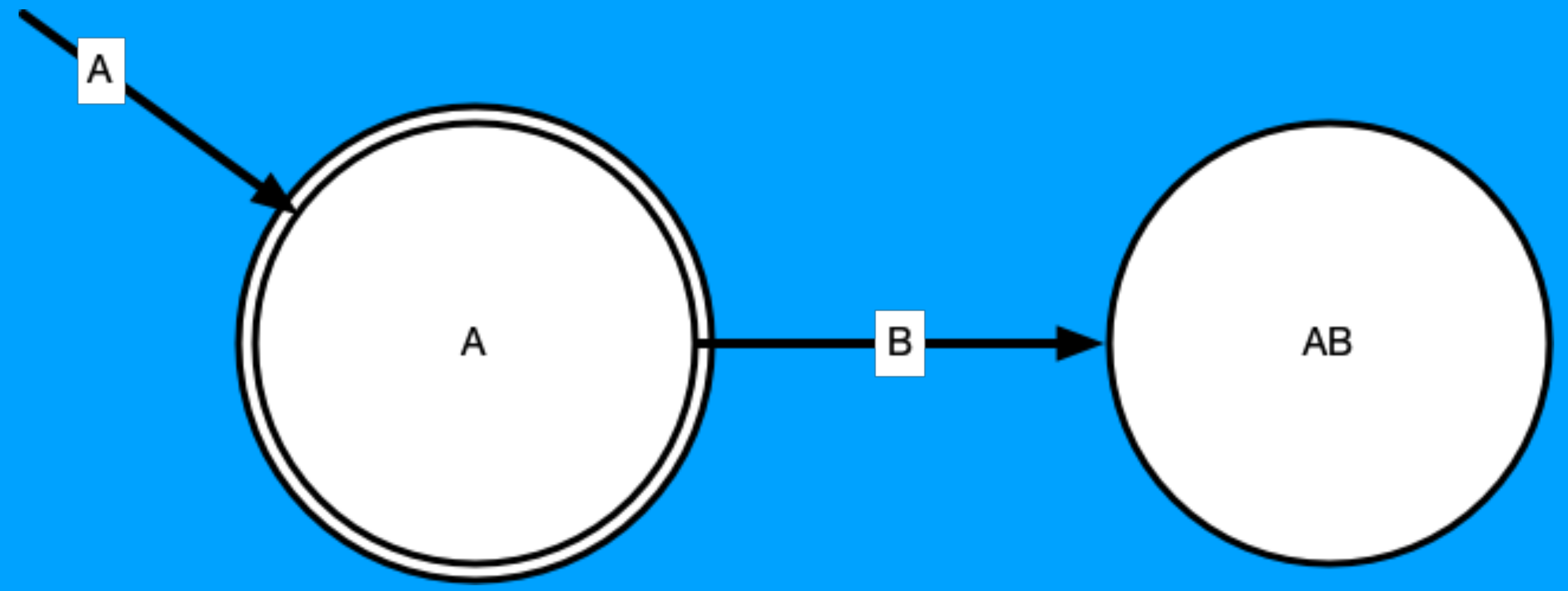
tw: @Erstejahre
email: wbrown@suse.de
https://github.com/Firstyear/purplecon_state_machines

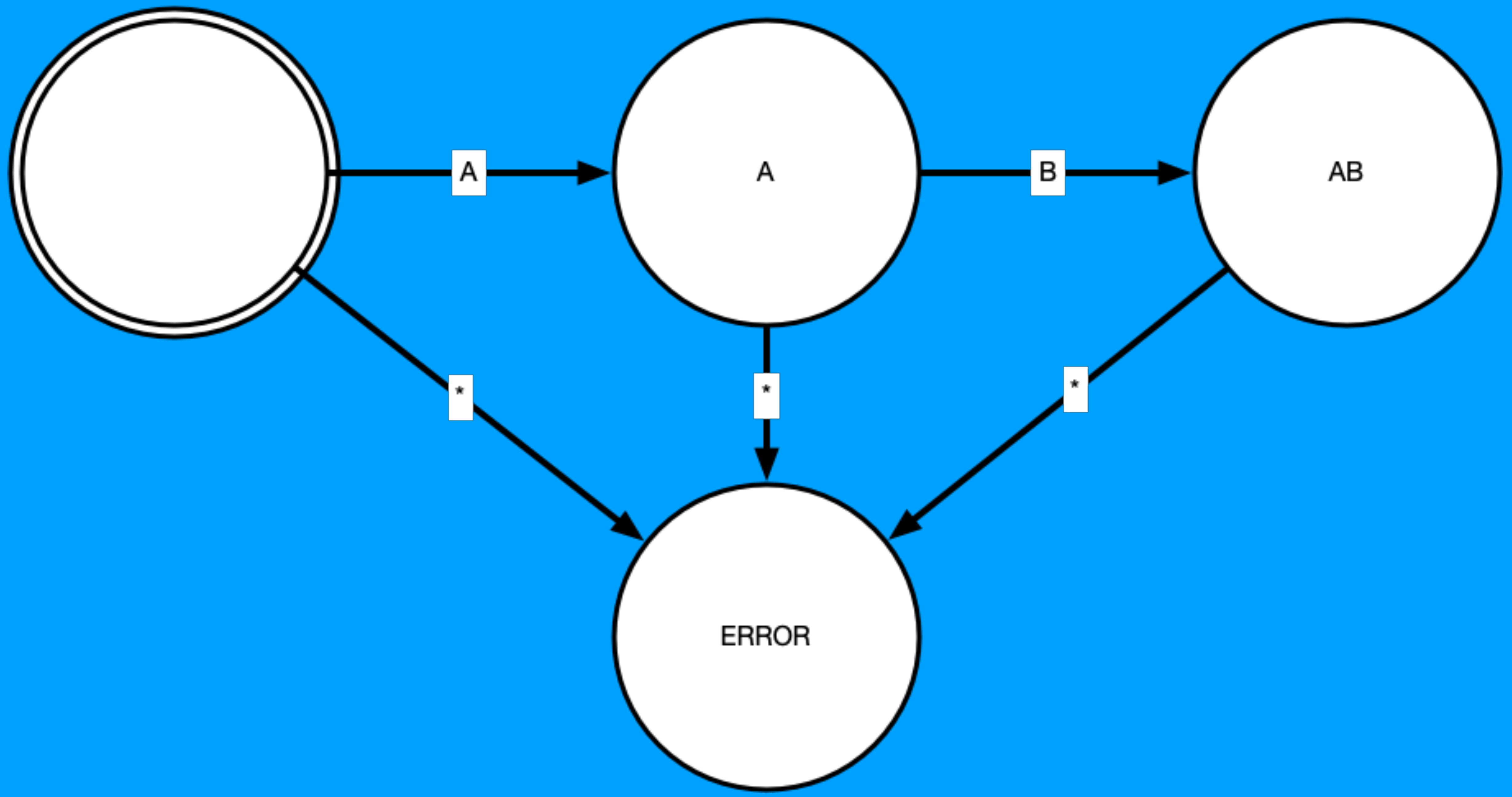
```
some_function(a: bool, b: Option<T>, c: usize) {
    // How many possible states?
}

some_function(a: bool, b: Option<T>, c: usize) {
    if !a || b.is_none() || c == 0 {
        return;
    }
    // do literally anything
}
```

tw: @Erstejahre
email: wbrown@suse.de
https://github.com/Firstyear/purplecon_state_machines

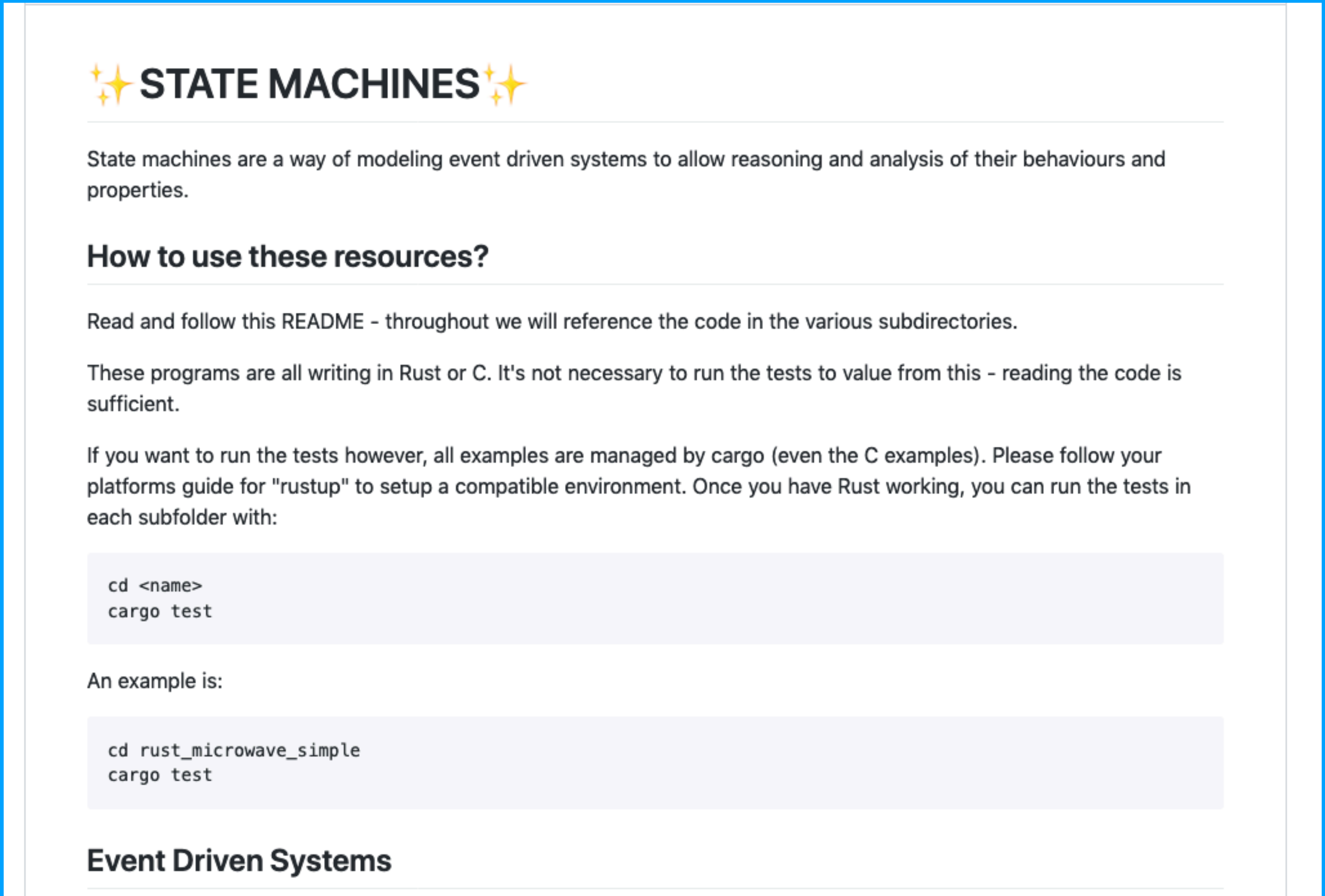
DFA or NFA?





tw: @Erstejahre
email: wbrown@suse.de
https://github.com/Firstyear/purplecon_state_machines

https://github.com/Firstyear/purplecon_state_machines



✨STATE MACHINES✨

State machines are a way of modeling event driven systems to allow reasoning and analysis of their behaviours and properties.

How to use these resources?

Read and follow this README - throughout we will reference the code in the various subdirectories.

These programs are all writing in Rust or C. It's not necessary to run the tests to value from this - reading the code is sufficient.

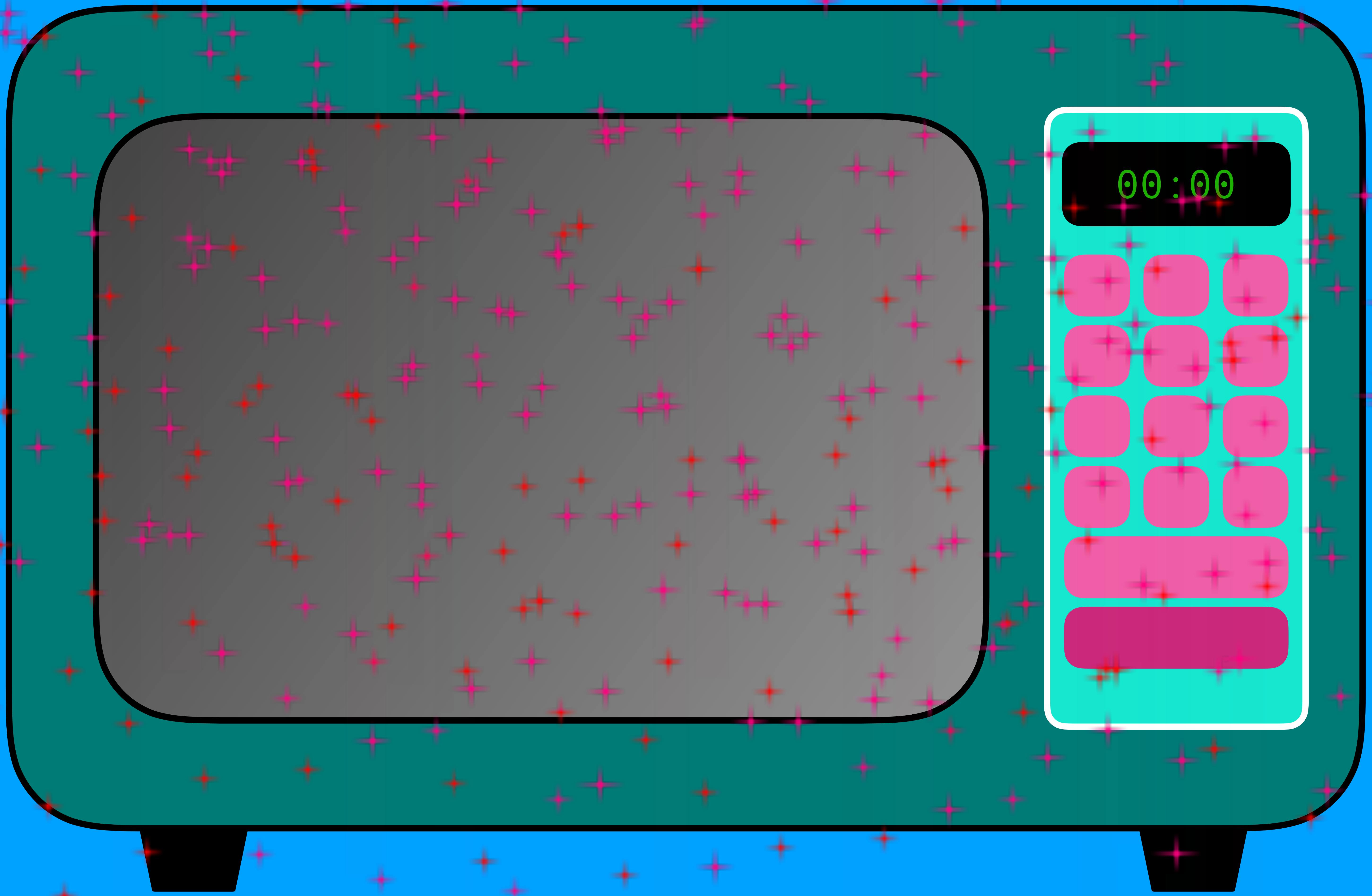
If you want to run the tests however, all examples are managed by cargo (even the C examples). Please follow your platforms guide for "rustup" to setup a compatible environment. Once you have Rust working, you can run the tests in each subfolder with:

```
cd <name>
cargo test
```

An example is:

```
cd rust_microwave_simple
cargo test
```

Event Driven Systems



tw: @Erstejahre

email: wbrown@suse.de

https://github.com/Firstyear/purplecon_state_machines