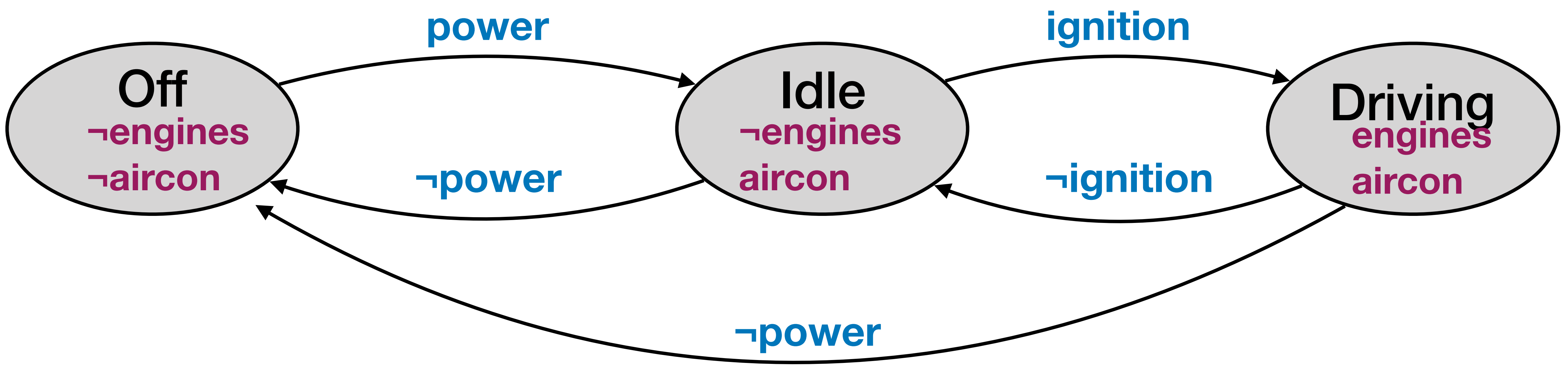
Pretending to verify a train controller with Lustre

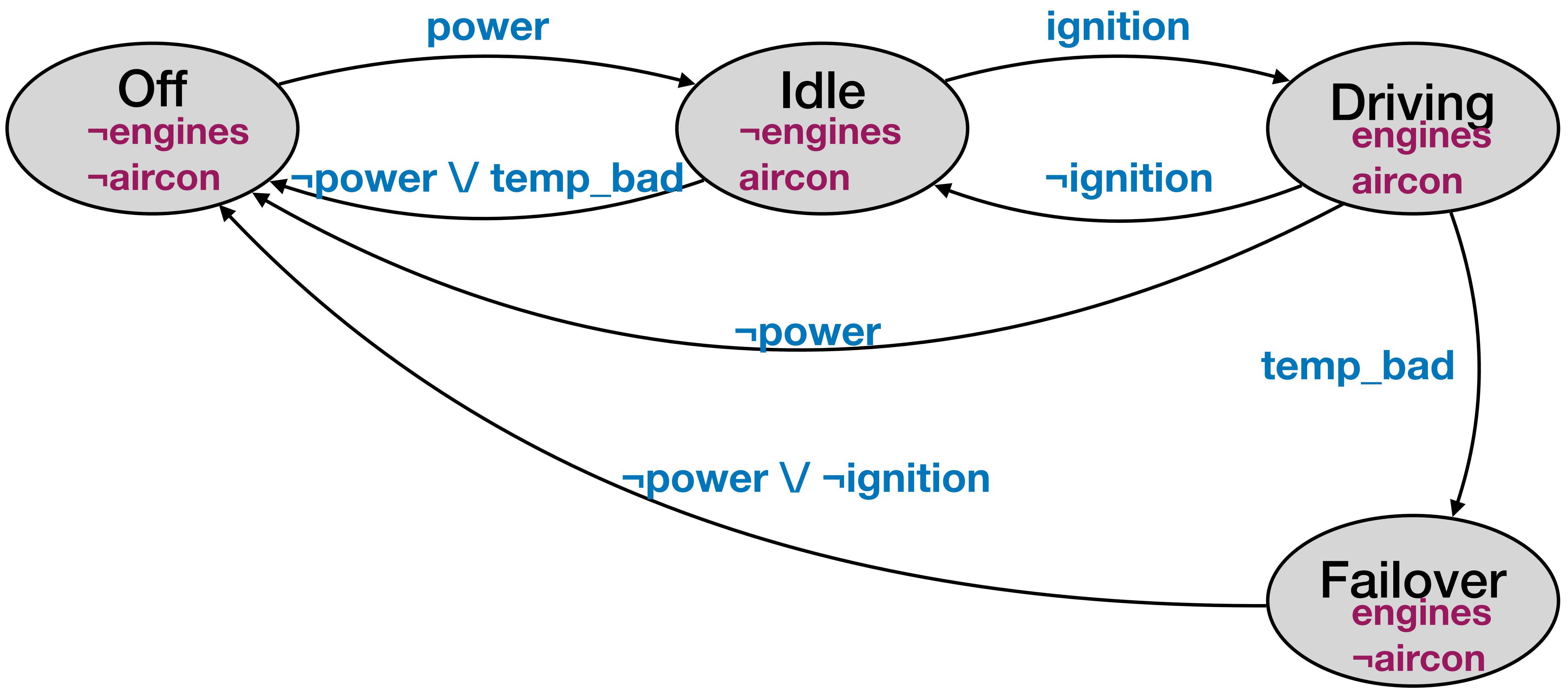
First, we need a pretend controller

- Controls the high level state machine
- Supervises the system

Simple controller



with failover mode



How do we implement this?

- Runs on microcontroller
- Limited space
- Can’t have out-of-memory errors
- Predictable runtime

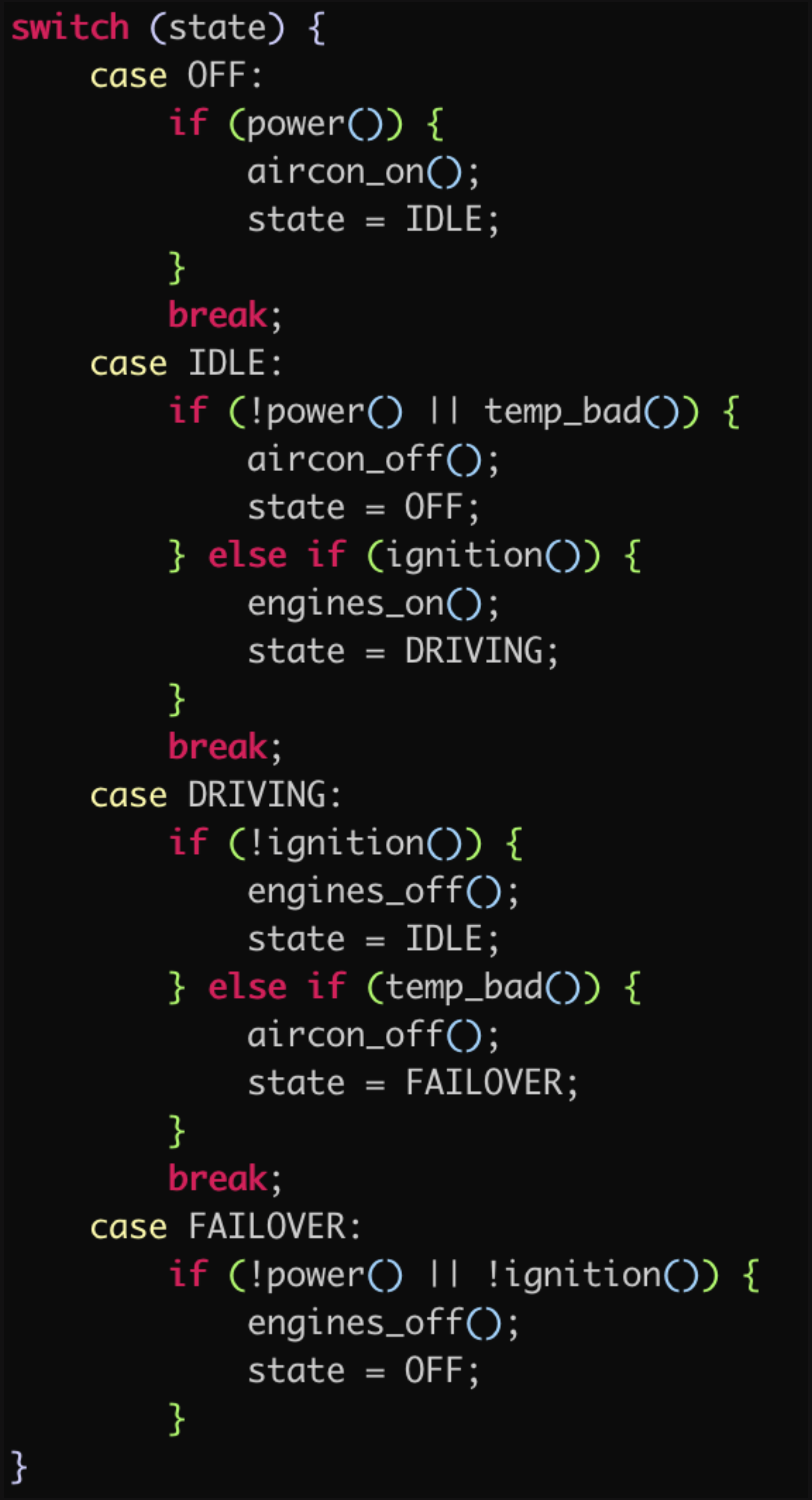
How do we implement this?

- Runs on microcontroller => C (of course)
- Limited space => C (no runtime)
- Can’t have out-of-memory errors => C (don’t malloc)
- Predictable runtime => C (don’t branch too much)

How do we implement this?

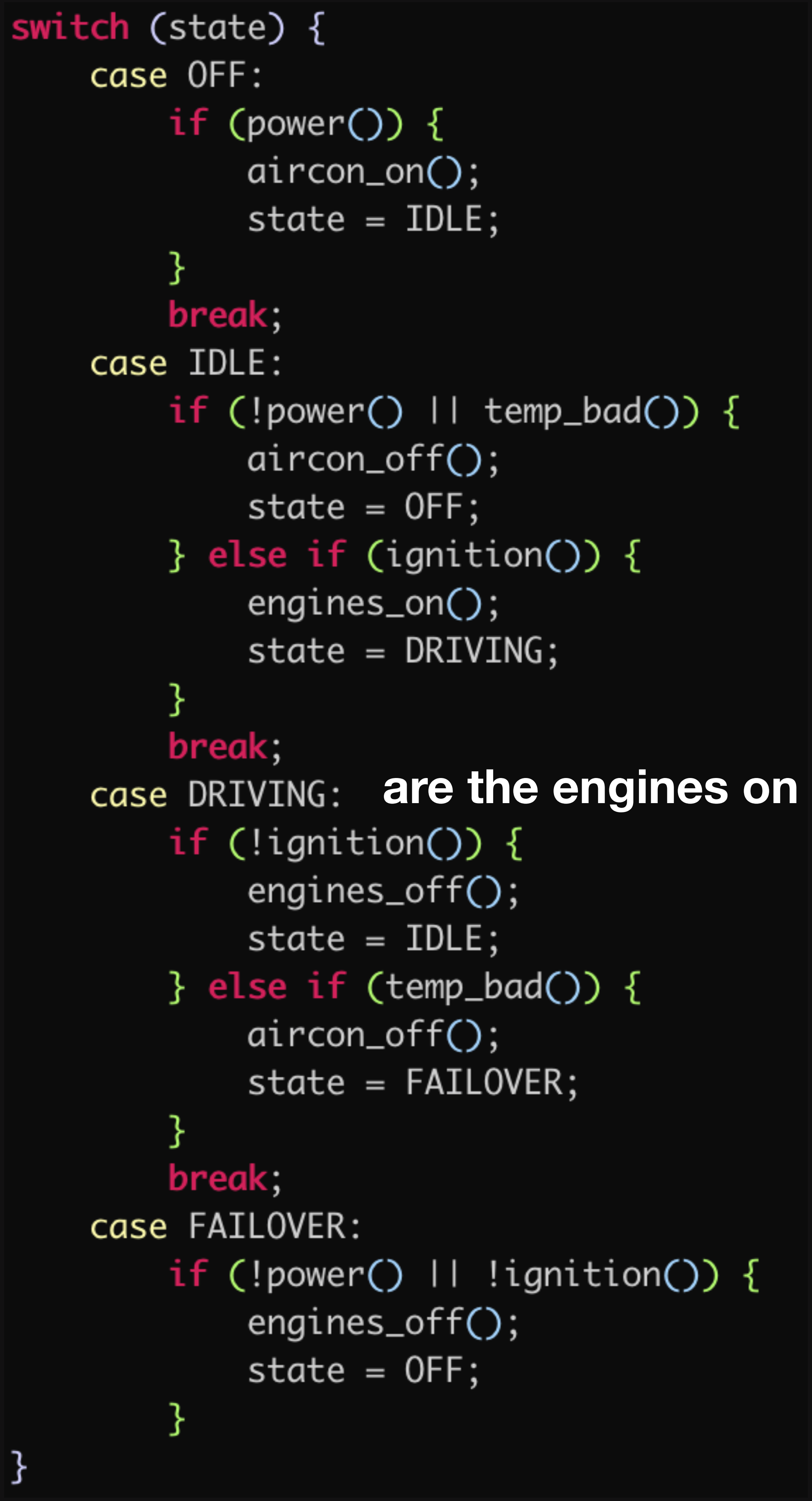
:’(

- Runs on microc ller => C (of c e)
- Limited space => C (no runtime)
- Can’t have out-o emory errors = (don’t malloc)
- Predictable runtime => C (don’t branc oo much)



```c
switch (state) {
    case OFF:
        if (power()) {
            aircon_on();
            state = IDLE;
        }
        break;
    case IDLE:
        if (!power() || temp_bad()) {
            aircon_off();
            state = OFF;
        } else if (ignition()) {
            engines_on();
            state = DRIVING;
        }
        break;
    case DRIVING:
        if (!ignition()) {
            engines_off();
            state = IDLE;
        } else if (temp_bad()) {
            aircon_off();
            state = FAILOVER;
        }
        break;
    case FAILOVER:
        if (!power() || !ignition()) {
            engines_off();
            state = OFF;
        }
}
```

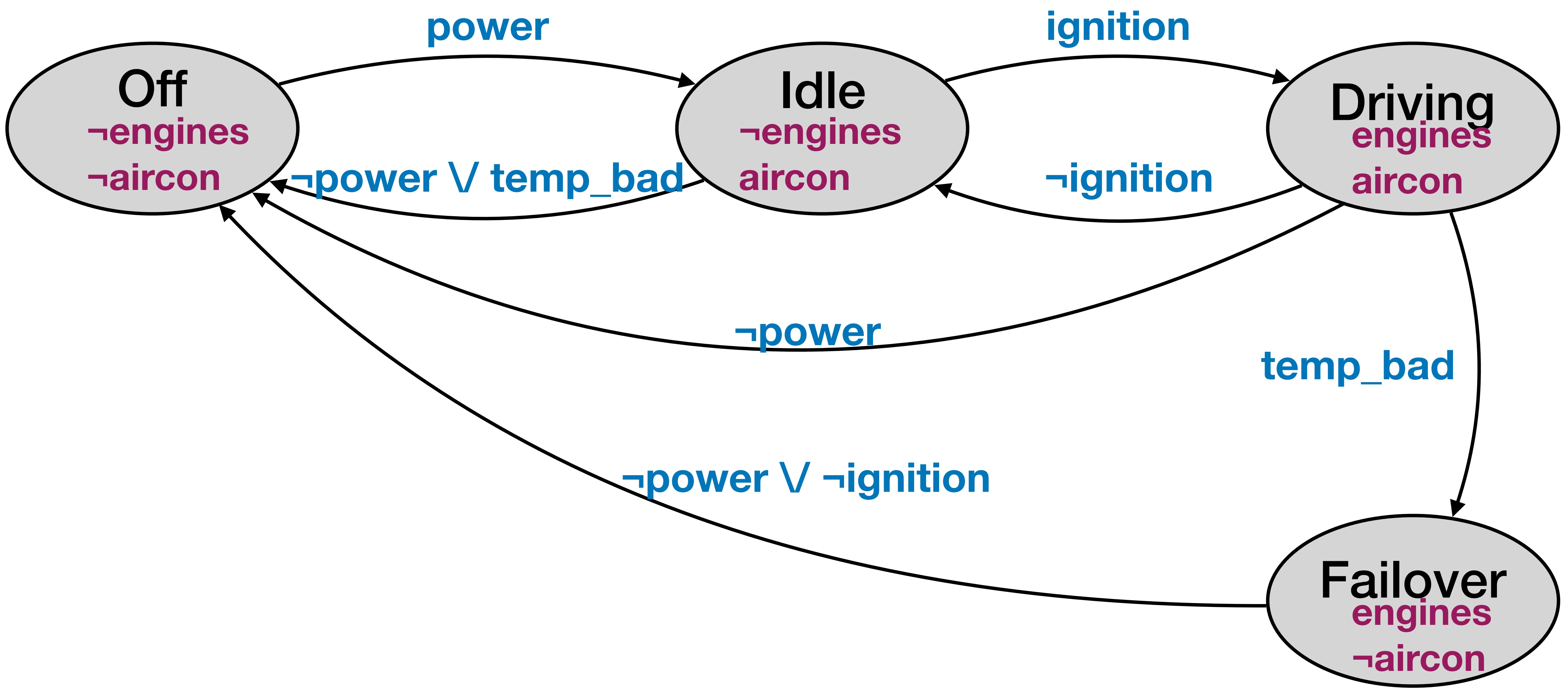
Bad



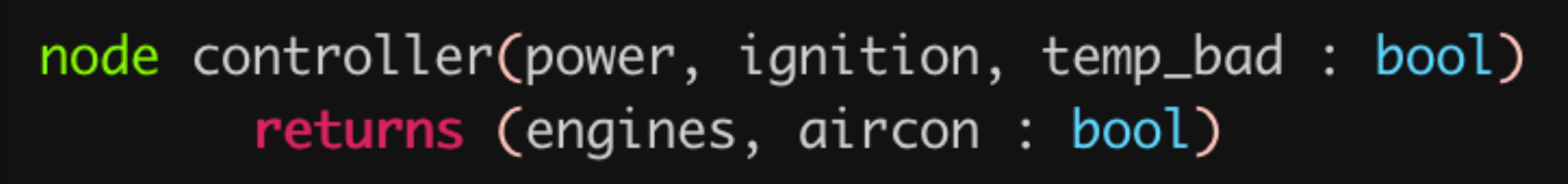
```
switch (state) {
    case OFF:
        if (power()) {
            aircon_on();
            state = IDLE;
        }
        break;
    case IDLE:
        if (!power() || temp_bad()) {
            aircon_off();
            state = OFF;
        } else if (ignition()) {
            engines_on();
            state = DRIVING;
        }
        break;
    case DRIVING:
        if (!ignition()) {
            engines_off();
            state = IDLE;
        } else if (temp_bad()) {
            aircon_off();
            state = FAILOVER;
        }
        break;
    case FAILOVER:
        if (!power() || !ignition()) {
            engines_off();
            state = OFF;
        }
}
```

are the engines on or off in DRIVING state?

Beautiful


power
ignition
Off
¬engines
¬aircon
Idle
¬engines
aircon
Driving
engines
aircon
¬power ⋁ temp_bad
¬ignition
¬power
temp_bad
¬power ⋁ ¬ignition
Failover
engines
¬aircon

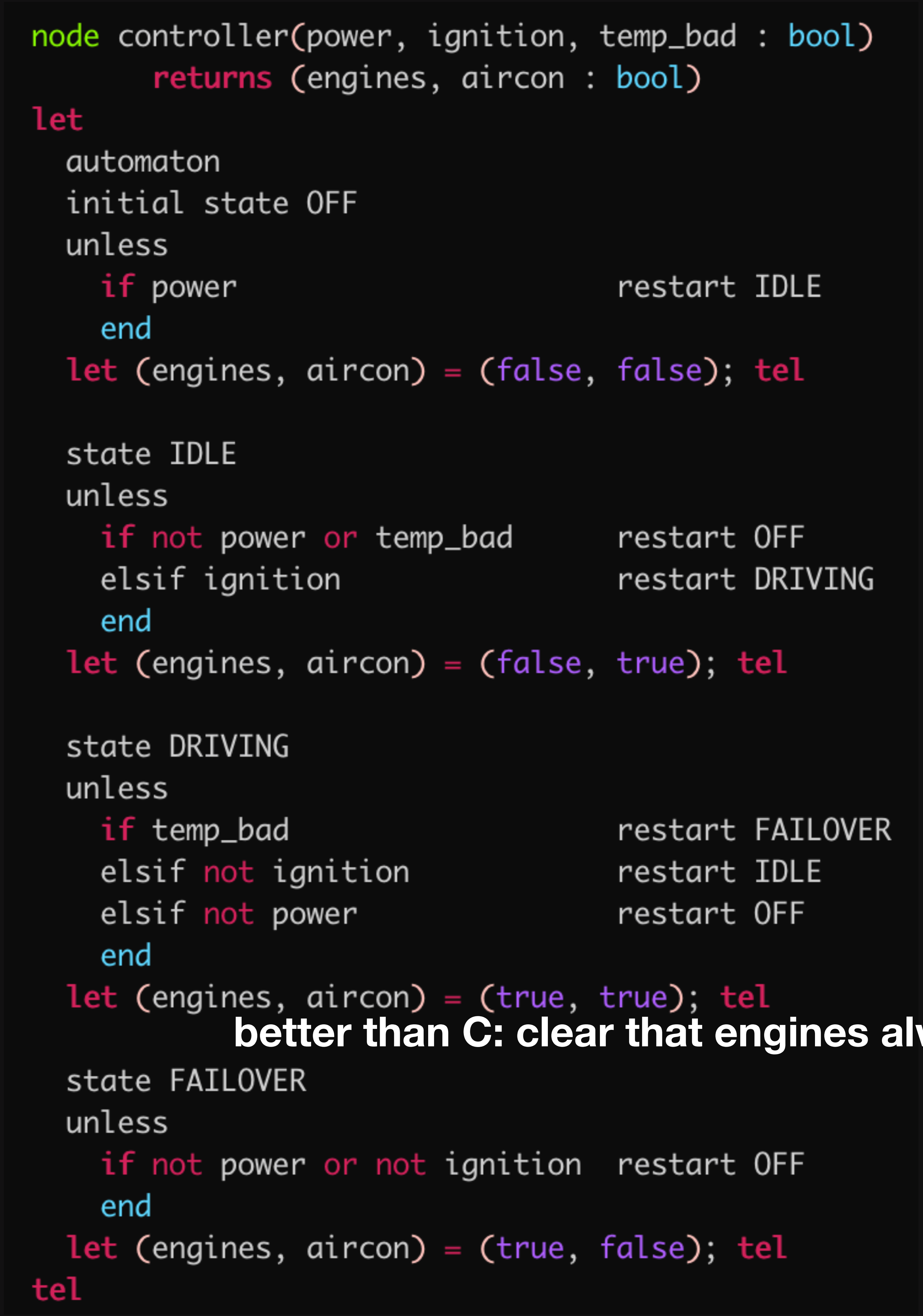
More beautiful



```
node controller(power, ignition, temp_bad : bool)
       returns (engines, aircon : bool)
```

```
node controller(power, ignition, temp_bad : bool)
       returns (engines, aircon : bool)
let
  automaton
  initial state OFF
  unless
    if power                      restart IDLE
    end
  let (engines, aircon) = (false, false); tel

  state IDLE
  unless
    if not power or temp_bad      restart OFF
    elsif ignition                restart DRIVING
    end
  let (engines, aircon) = (false, true); tel

  state DRIVING
  unless
    if temp_bad                   restart FAILOVER
    elsif not ignition            restart IDLE
    elsif not power               restart OFF
    end
  let (engines, aircon) = (true, true); tel

  state FAILOVER
  unless
    if not power or not ignition  restart OFF
    end
  let (engines, aircon) = (true, false); tel
tel
```

Good



```
node controller(power, ignition, temp_bad : bool)
       returns (engines, aircon : bool)
let
  automaton
  initial state OFF
  unless
    if power                          restart IDLE
    end
  let (engines, aircon) = (false, false); tel

  state IDLE
  unless
    if not power or temp_bad          restart OFF
    elsif ignition                    restart DRIVING
    end
  let (engines, aircon) = (false, true); tel

  state DRIVING
  unless
    if temp_bad                       restart FAILOVER
    elsif not ignition                restart IDLE
    elsif not power                   restart OFF
    end
  let (engines, aircon) = (true, true); tel

  state FAILOVER
  unless
    if not power or not ignition      restart OFF
    end
  let (engines, aircon) = (true, false); tel
tel
```

better than C: clear that engines always on in DRIVING

Good

```
node controller(power, ignition, temp_bad : bool)
       returns (engines, aircon : bool)
let
  automaton
  initial state OFF
  unless
    if power                              restart IDLE
    end
  let (engines, aircon) = (false, false); tel

  state IDLE
  unless
    if not power or temp_bad              restart OFF
    elsif ignition                        restart DRIVING
    end
  let (engines, aircon) = (false, true); tel

  state DRIVING
  unless
```

better than diagram: unambiguous transition precedence

```
    if temp_bad                           restart FAILOVER
    elsif not ignition                    restart IDLE
    elsif not power                       restart OFF
    end
  let (engines, aircon) = (true, true); tel
```

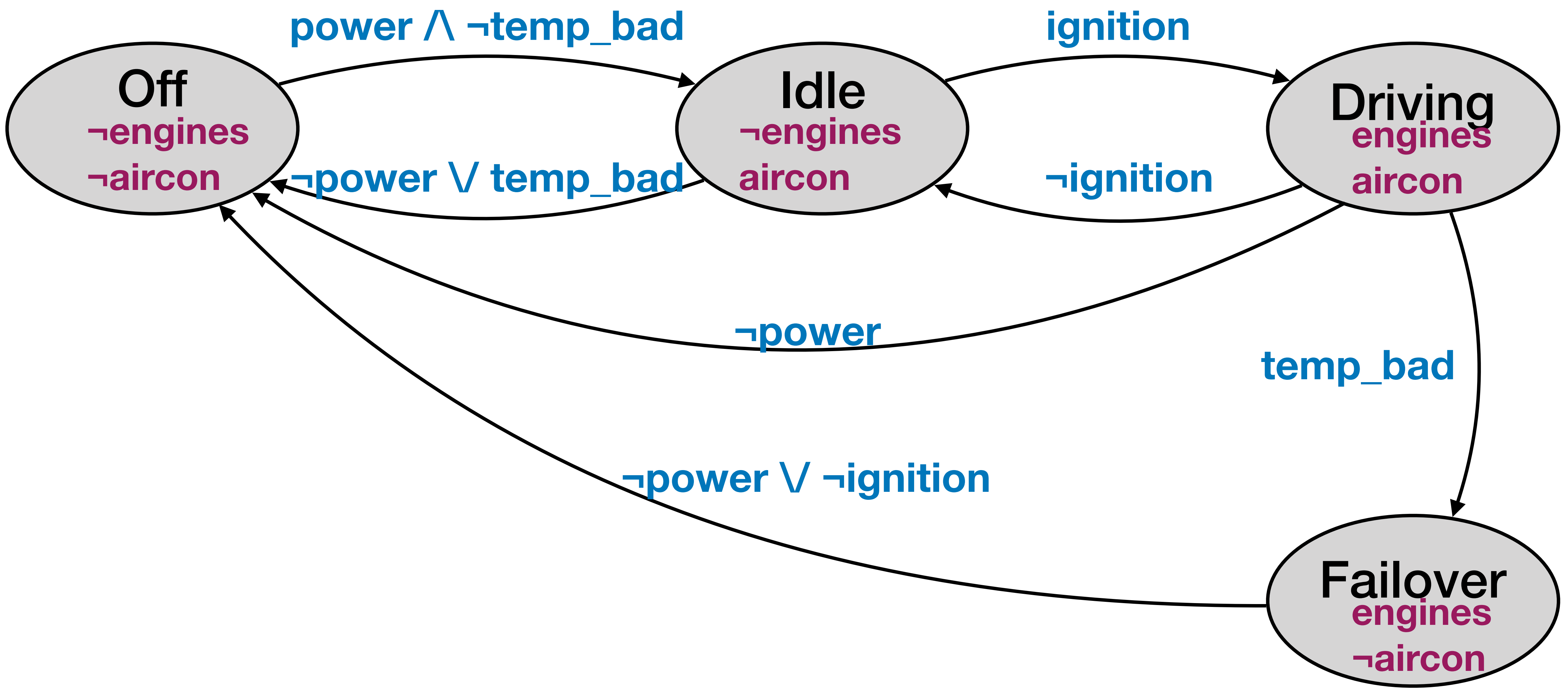
better than C: clear that engines always on in DRIVING

```
  state FAILOVER
  unless
    if not power or not ignition          restart OFF
    end
  let (engines, aircon) = (true, false); tel
tel
```

Workflow

- Write Lustre
- Verify Lustre implementation of FSM
- Compile to C
- Integrate generated C with rest of system

A property we might want: “if temp is bad, air con is off”

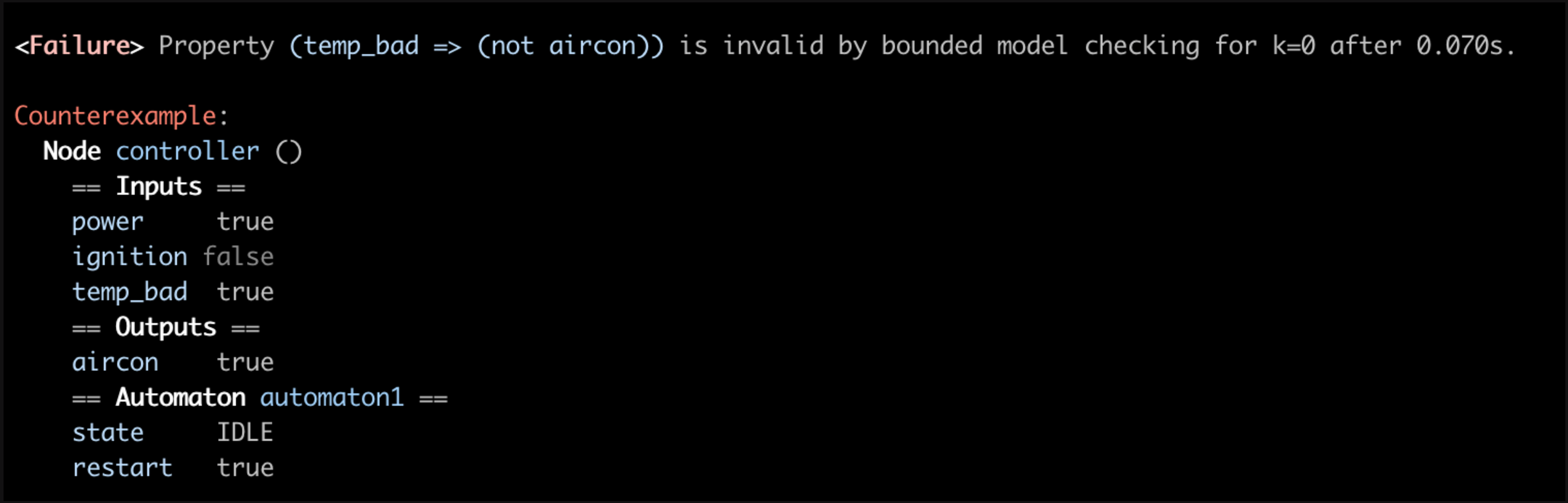


Checking properties with Kind2

```
node controller(power, ignition, temp_bad : bool)
       returns (engines, aircon : bool)
let
  --%PROPERTY temp_bad => not aircon;
```

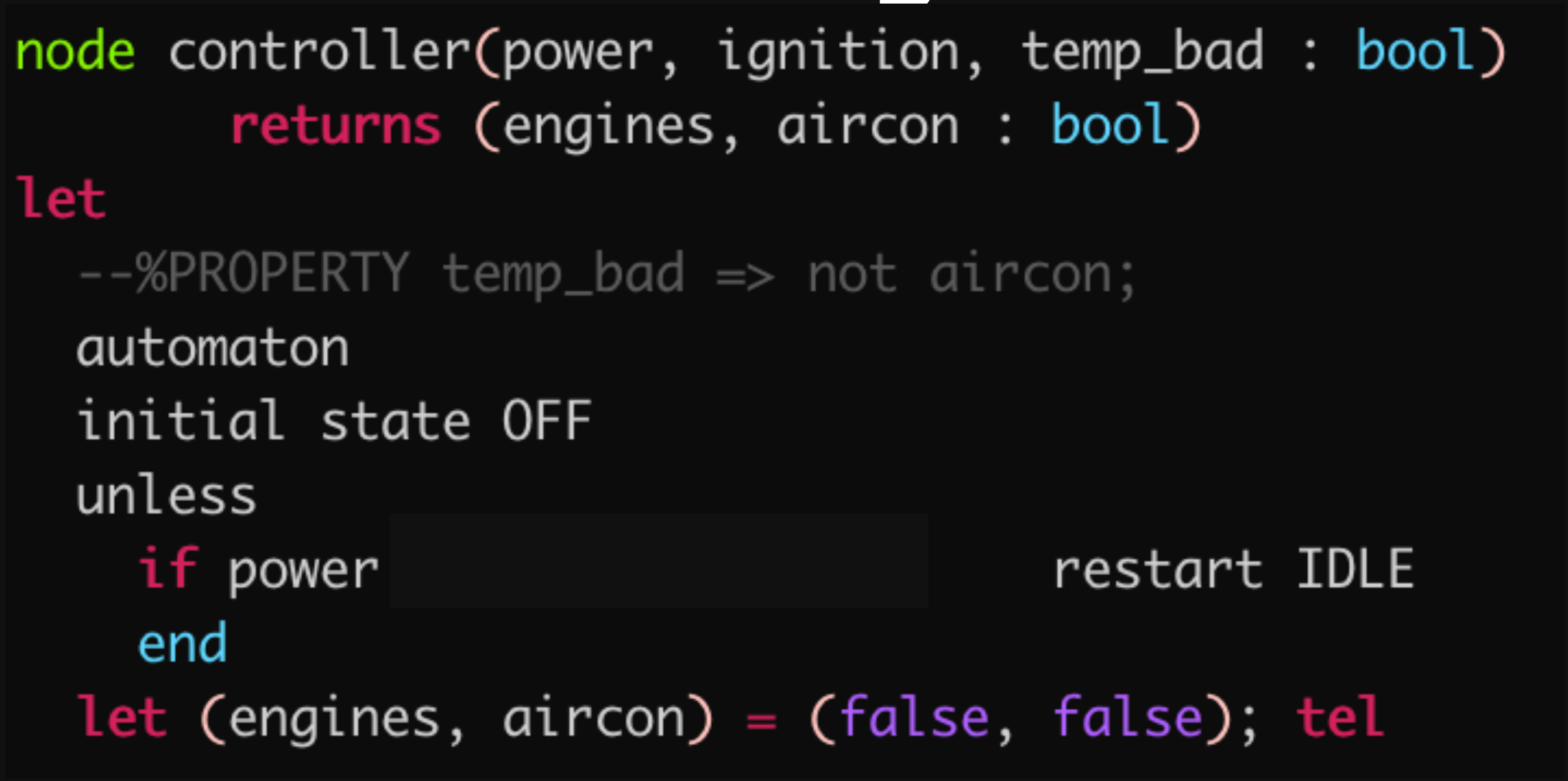
Checking properties with Kind2

```
node controller(power, ignition, temp_bad : bool)
     returns (engines, aircon : bool)
let
  --%PROPERTY temp_bad => not aircon;
```



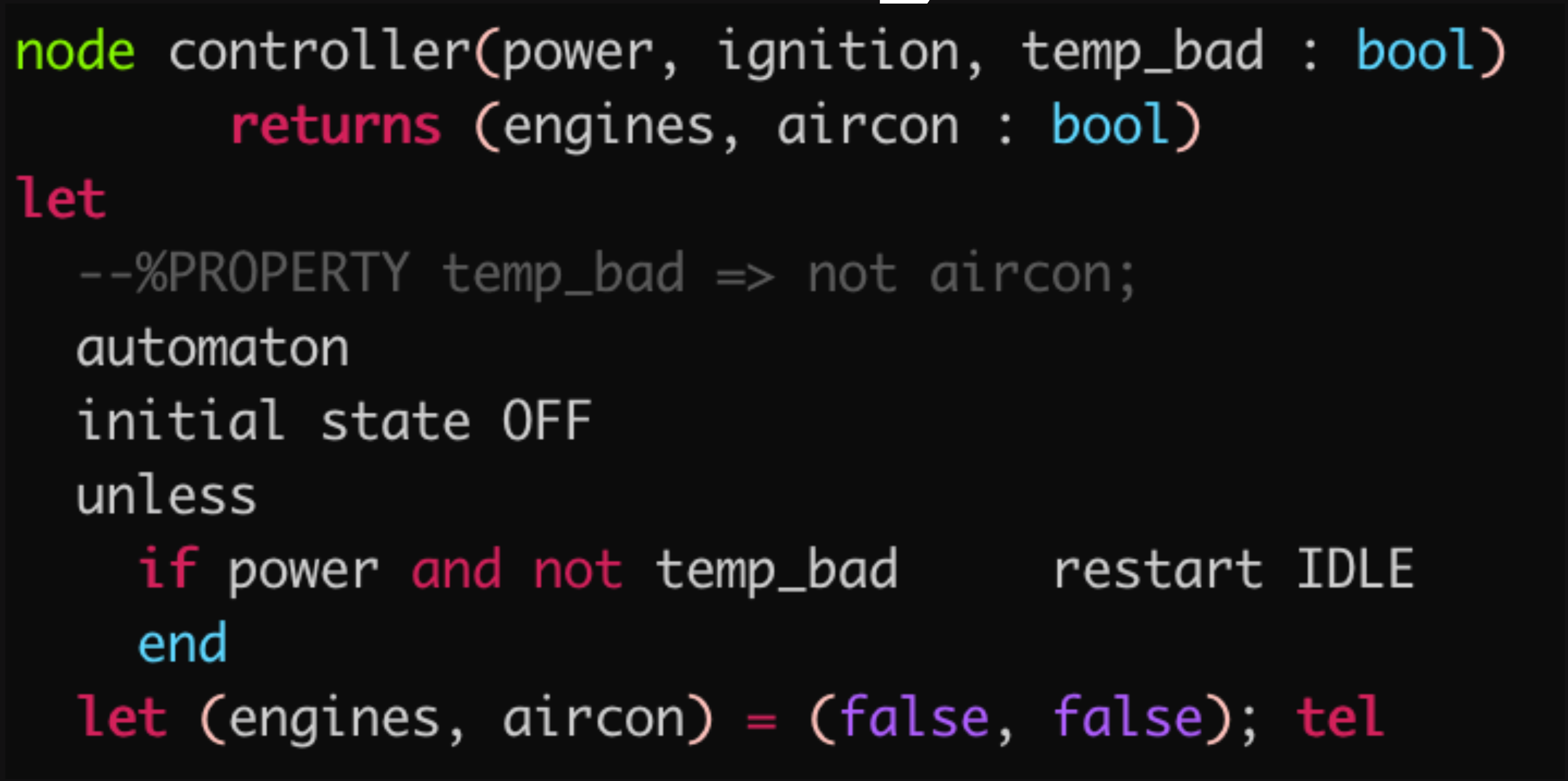
```
<Failure> Property (temp_bad => (not aircon)) is invalid by bounded model checking for k=0 after 0.070s.

Counterexample:
  Node controller ()
    == Inputs ==
    power    true
    ignition false
    temp_bad true
    == Outputs ==
    aircon   true
    == Automaton automaton1 ==
    state    IDLE
    restart  true
```

Fix is easy...



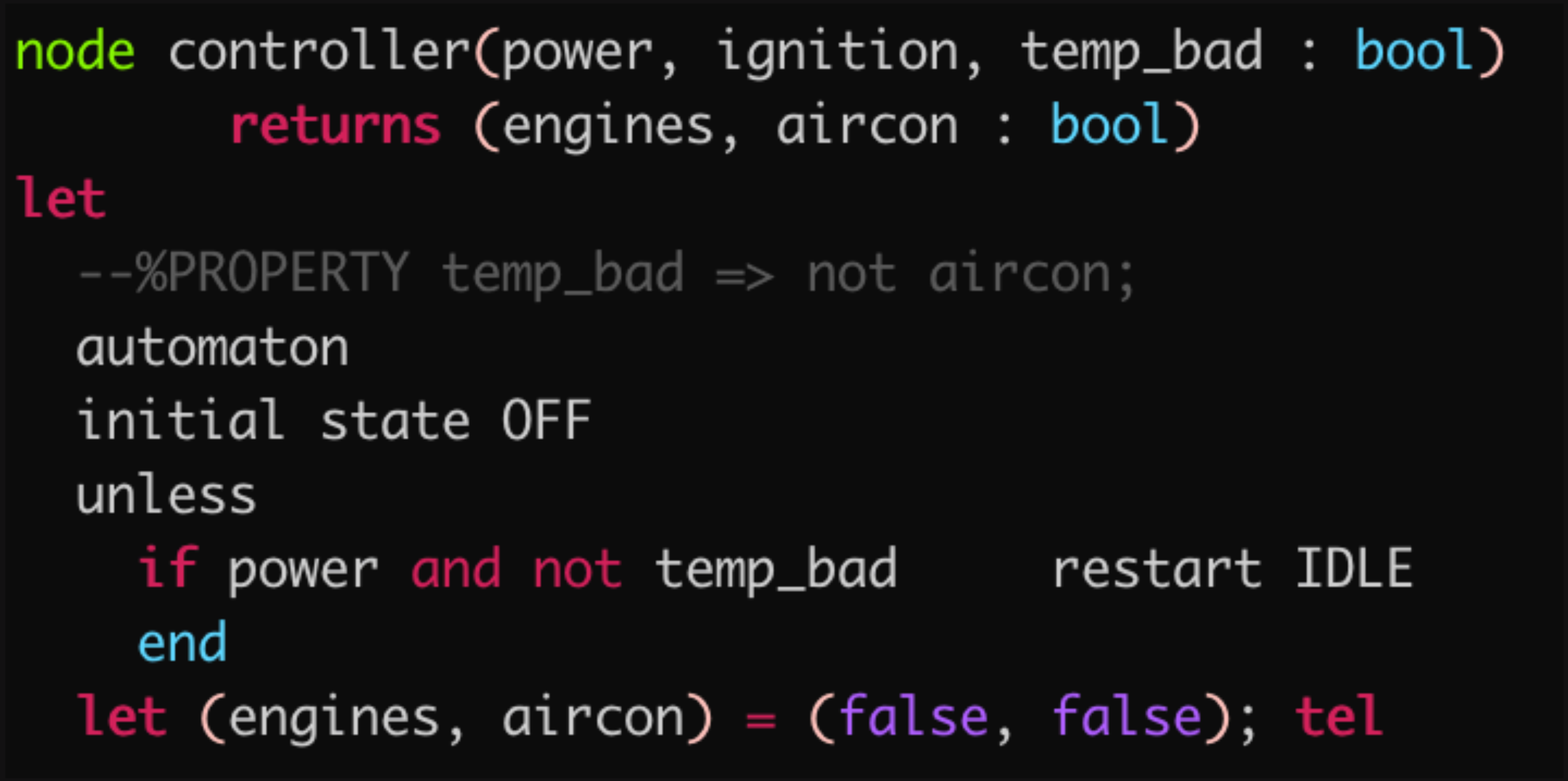
```
node controller(power, ignition, temp_bad : bool)
       returns (engines, aircon : bool)
let
  --%PROPERTY temp_bad => not aircon;
  automaton
  initial state OFF
  unless
    if power                         restart IDLE
    end
  let (engines, aircon) = (false, false); tel
```

Fix is easy...

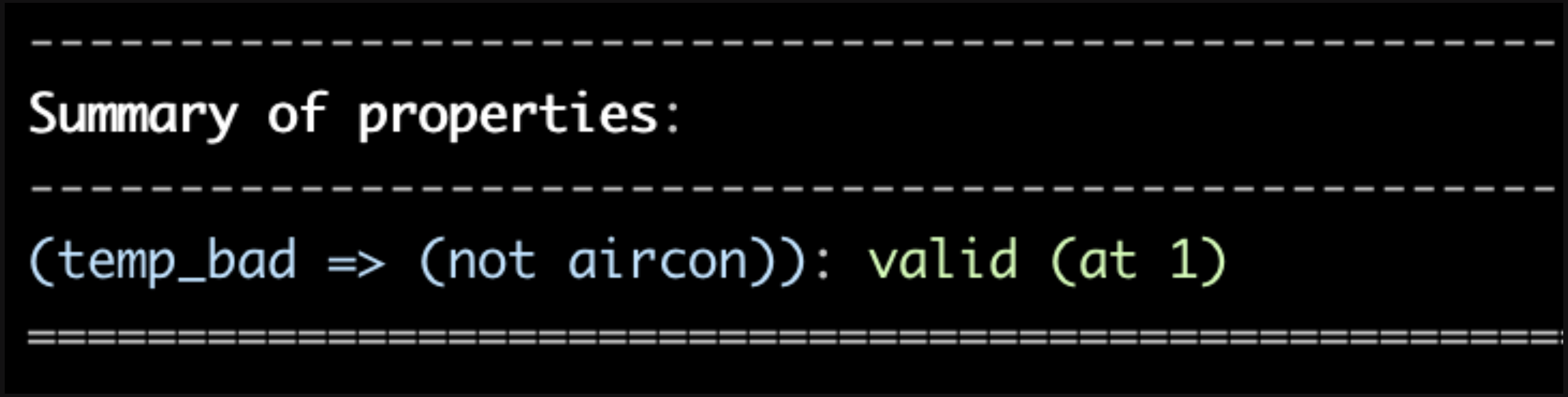


```
node controller(power, ignition, temp_bad : bool)
       returns (engines, aircon : bool)
let
  --%PROPERTY temp_bad => not aircon;
  automaton
  initial state OFF
  unless
    if power and not temp_bad     restart IDLE
    end
  let (engines, aircon) = (false, false); tel
```

Success!



```
node controller(power, ignition, temp_bad : bool)
       returns (engines, aircon : bool)
let
  --%PROPERTY temp_bad => not aircon;
  automaton
  initial state OFF
  unless
    if power and not temp_bad     restart IDLE
    end
  let (engines, aircon) = (false, false); tel
```



```
-----------------------------------------------
Summary of properties:
-----------------------------------------------
(temp_bad => (not aircon)): valid (at 1)
===============================================
```

Conclusion

- It’s a simple idea, but it fills a niche that I don’t know of any other solutions for.
- I think Lustre is a language that really nails the “less expressive is better” as it allows strong reasoning about embedded code