# TLA+

## And how to use it to catch bugs

Pepe García pepe@goodnotes.com

# Plan for today

- Why model systems?
- What is TLA+?
- How to use TLA+?
- Use in the real world

# Glossary

- **System**: the thing we're trying to model.
- **Model**: a description of a system.
- **State space**: the set of all possible states of a system.
- **State**: the snapshot of the system at a given point in time.
- **Behaviour**: a *path* in the state graph

# Why model systems?

We model systems to undertand them better, and to be able to proof certain properties of them.

# Why model systems?

Remember, the map is not the territory. We're trying to understand the territory, not the map.

When modelling our system and proving properties of it, we're proving properties of the model, which should map to our system correctly.

# Why model systems?

**Warning**

With TLA+, we're gonna write code to model our system, but we won't be able to use this code to generate code for our system.

# Why model systems?

**Most Frequently Asked Question**

On learning about TLA+, engineers usually ask, "How do we know that the executable code correctly implements the verified design?" The answer is we do not know. Despite this, formal methods help in multiple ways:

*Get design right*. Formal methods help engineers get the design right, which is a necessary first step toward getting the code right. If the design is broken, then the code is almost certainly broken. Engineers are unlikely to realize the design is incorrect while focused on coding;

*Gain better understanding*. Formal methods help engineers gain a better understanding of the design. Improved understanding can only increase the chances they will get the code right; and

*Write better code*. Formal methods can help engineers write better "self-diagnosing code" in the form of assertions. Formal methods can help improve assertions that help improve the quality of code.

From Lamport's comment on Amazon's paper[1].

[1] https://lamport.azurewebsites.net/tla/amazon-excerpt.html

# What is TLA+?

- TLA+ is a formal specification language
- invented by Leslie Lamport (Turing award 2013, LaTeX, Paxos, Lamport Clocks, ...)
- used to model concurrent systems

```
VARIABLE clock

Init == clock \in {0, 1}

Tick == IF clock = 0 THEN clock' = 1 ELSE clock' = 0

Spec == Init /\ [][Tick]_<<clock>>
```

# What's PlusCal?

- PlusCal is a high-level language that compiles to TLA+
- It's easier to write, and closer to imperative languages

```
-- fair algorithm OneBitClock {
  variable clock \in {0, 1};
  {
    while (TRUE) {
      if (clock = 0)
        clock := 1
      else
        clock := 0
    }
  }
}
```

# What's PlusCal?

- PlusCal is a high-level language that compiles to TLA+
- It's easier to write, and closer to imperative languages

```
-- fair algorithm OneBitClock {
  variable clock \in {0, 1};
  {
    while (TRUE) {
      if (clock = 0)
        clock := 1
      else
        clock := 0
    }
  }
}
```

**We can think of TLA+ as the assembly specification language and PlusCal as C, that compiles to it**

# Use in the real world

- AWS[2]
- Microsoft[3]
- Oracle[4]
- ...
- GoodNotes

[2] https://awsmaniac.com/how-formal-methods-helped-aws-to-design-amazing-services/

[3] https://www.microsoft.com/en-us/research/blog/tla-foundation-aims-to-bring-math-based-software-modeling-to-the-mainstream/

[4] https://blogs.oracle.com/cloud-infrastructure/post/sleeping-soundly-with-the-help-of-tla

# Quick intro to PlusCal

- `while`, `if`... have the same semantics as in any other language
- Lists are delimited by << and >> and comma separated.
- Sets are enclosed by { and }, and comma separated.
- Dictionaries are enclosed by [ and ], and are comma separated. Keys and values are mapped via |->.
- Functions are functions in the mathematical way, mapping between a domain set and a codomain set. The closest to functions in other languages are operators.
- We create processes to denote independent processes in our system.

# Quick intro to PlusCal

Let's demo it!

Let's create a model for a simple system: A wire transfer with overdraft protection.

Open `overdraft.tla`.

# Quick intro to Pluscal

Remember that PlusCal transpiles to TLA+, we need to first convert it to TLA+.

And then, run TLC in the compiled model.

# Real world example

At work, we try to save money by moving events from our event storage from a *hot* storage to a *cold* storage. This is, from a relational DB, to S3.

Sometimes, the number of events for a single document is huge, and restoring it takes a long time. Initially, our algorithm for restoring a *cold* stream of document events was something like this in pseudocode:

```
def restore_events(event_stream_id):
    restored = False
    while not restored:
        try:
            enqueue_restoration_request(event_stream_id)
            # timeout in 60 secs
            wait_for_notification(event_stream_id, 60)
            restored = True
        catch TimeoutException:
            continue
```

# Real world example

Of course, we have a consumer counterpart:

```
class RestorationRequestConsumer:
    def consume(event):
        restore_events_to_hot_storage(event.stream_id)

        notify_restoration_done(event.stream_id)
```

# Real world example

You can see that we have a producer, and a consumer. The producer will wait for a minute for a notification that the consumer finished working. If it doesn't get the notification, it will enqueue again.

The way we were autoscaling this particular piece didn't have the *queue pressure* into consideration to add more consumers, and we wound up having an incident in this subsystem, because the producer was publishing more and more events even though the consumer wasn't able to keep up.

The problematic part for us was to just wait for a minute and then retry.

# Real world example. Solution

Let's write this in Pluscal and see if we can model it to find the error.

# Real world example. Solution

Head to `0_event_restoration.tla`.

# Real world example. Solution

After running it, you can see that we're encountering an error. Our invariant is violated, meaning that we're enqueuing the same event twice.

This is great, we've managed to reproduce the error in our model, but we gotta find a solution now.

# Real world example. Solution

Now, on the consumer side, we will update the restoration time when we start working on an event.

```
class RestorationRequestConsumer:
    def consume(event):
        update_restore_time(event, now())
        restore_events_to_hot_storage(event.stream_id)

        notify_restoration_done(event.stream_id)
```

# Real world example. Solution

And, we'll update the producer to check if the restoration time is too old, and only then start working on it.

```
def restore_events(event_stream_id):
    restored = False
    while not restored:
        try:
            enqueue_restoration_request(event_stream_id)
            # timeout in 60 secs
            wait_for_notification(event_stream_id, 60)
            restored = True
        catch TimeoutException:
            if is_restoration_too_old(event_stream_id):
                continue
            else:
                wait(60)
                continue
```

# Real world example. Solution

Head over to `1_fixing_invariant.tla`.

# Thanks

Thank to my **amazing colleages** that didn't need formal verification to understand this, and helped me with actual examples of how the previous model could fail, and be fixed.

Find the slides at https://github.com/pepegar/courses-and-presentations/blob/main/slides/presentations/tla/tla-intro.pdf