Introduction to TRULY AGILE System Design

Vesal Vojdani
University of Tartu
### Why TLA+

Blame Amazon…

<table>
<thead>
<tr>
<th>System</th>
<th>Components</th>
<th>Line Count (Excluding Comments)</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>S3</td>
<td>Fault-tolerant, low-level network algorithm</td>
<td>804 PlusCal</td>
<td>Found two bugs, then others in proposed optimizations</td>
</tr>
<tr>
<td></td>
<td>Background redistribution of data</td>
<td>645 PlusCal</td>
<td>Found one bug, then another in the first proposed fix</td>
</tr>
<tr>
<td>DynamoDB</td>
<td>Replication and group-membership system</td>
<td>939 TLA+</td>
<td>Found three bugs requiring traces of up to 35 steps</td>
</tr>
<tr>
<td>EBS</td>
<td>Volume management</td>
<td>102 PlusCal</td>
<td>Found three bugs</td>
</tr>
<tr>
<td>Internal distributed</td>
<td>Lock-free data structure</td>
<td>223 PlusCal</td>
<td>Improved confidence though failed to find a liveness bug, as liveness</td>
</tr>
<tr>
<td>lock manager</td>
<td>Fault-tolerant replication-and-reconfiguration algorithm</td>
<td>318 TLA+</td>
<td>not checked</td>
</tr>
</tbody>
</table>
Concurrent Systems

- TLA+ has mainly been used to validate complicated distributed algorithms, mainly commitment protocols.
- It was developed by Leslie Lamport who innovated many such algorithms.
- It has found continued use in cloud storage teams at Microsoft and Amazon.
- What about you?
What we care about...

- Get a taste of truly agile development: debuggable design!

- What can go wrong in a concurrent system (race conditions, deadlocks, starvation, …)

- Understand the vocabulary of formal verification (safety and liveness, invariants, counter-example traces)
Hello, Concurrency!

Classic Example of a data race
Time-of-check to time-of-use
Study Information System

- Teachers may create a protocol and select students.

- Having selected a bunch of them, he can finalize the protocol.

- Very simple, but… we don’t want a student to be added to two different protocols.
Demo: SIS 1.0
Model of creating exam protocols.
Safety Condition

No student should be added to multiple protocols
Common Idiom in Logic

When we need to talk about two different elements, we say:

Any two protocols overlap only if they are the same.
Of course, this is violated...

Two protocols can just add the same people
process createprotocol \in Protocols
variable selected = {}
begin
  SelectUsers:
(* create protocol and select students: *)
  with s \in \textbf{SUBSET} free do selected := s end with;
  LockThemUp:
  locked := locked \cup selected;
Decide:
either
(* save protocol *)
  graded_students[self] := selected
or
(* leave open, have coffee, close browser *)
skip
end either
end process

Add some locking!
This is almost right… (pedantic point)
Add some locking!

Locking needs to be atomic.
And now?

- We run the model checker.
- It seems to work. So we are done.
- Really, we are actually done now.
- This is the SIS 1.0 implementation of exam protocols!
- Whenever we add synchronization to deal with a safety feature, you should ask…
What about liveness?

• Let’s say all free students are added to the last protocol by Ülle Holm…

• Will this allow us to eventually reach a state where all students have been added to a protocol?

\[
All\text{Graded} \triangleq \Diamond (\forall s \in Students : \exists p \in Protocols : (s \in graded\_students[p]))
\]

• Can something go wrong?

```plaintext
process finalproto = "finally"
begin
Finally:
    when \forall p \in Protocols : pc[p] = "Done";
    with p \in Protocols do
        graded\_students[p] := graded\_students[p] \cup free
    end with
end process
```
Of course, we need to unlock!
It’s so obvious when you model things…
The actual solution at our university

Ülle Holm would manually have to clean this mess.

“Dear Vesal, please just don’t create any protocols on your own at all. Send me an e-mail instead!”
What is more cost-effective: TLA or Ülle Holm.

Your answer will be considered correct if you say “TLA is more cost-effective for the consumer; Ülle Holm is cheaper for the developers.”

You may also (but don’t have to) state your own opinion. Honesty will not be punished (“TLA is unrealistic …”) As long as you remain professional (“TLA sucks!”)

(This is not a joke. The affective domain of learning, i.e., your attitudes, are important.)
Moral of the story

• TLA may be difficult, logic may be difficult, but… building reliable systems is even harder!

• “System’s hard; must work harder!”

• If concurrency is an afterthought, this often happens: add locking to deal with safety, only to introduce a liveness problem.

• TLA+ gives you the agility to play around with your design to identify these subtle issues before putting it into code.