GOSIM®
GLOBAL OPEN-SOURCE INNOVATION MEETUP

GOSIM CHINA 2024

Taming concurrency with (in-)formal methods in Servo

How TLA+ is used as part of the toolbox to build Servo, a Web engine.

GOSIM
GLOBAL OPEN-SOURCE INNOVATION MEETUP

GOSIM CHINA 2024

Gregory Terzian
Servo TSC member

目录
Contents

What is Servo?

GOSIM®
GLOBAL OPEN-SOURCE INNOVATION MEETUP

servo

The embeddable, independent, memory-safe, modular, parallel web browser engine

What is Servo?

GOSIM®
GLOBAL OPEN-SOURCE INNOVATION MEETUP

Web engine = that which runs the the Web application platform.

Application(s)	Safari, Wechat, …
Application Platform(s)	Web, Native, Mini-apps Web: Webkit, Servo, …
Operating System	

A Web Engine For The Future 🔮

Highlights ✨

- **Independent:** Open governance
- **Performant:** Web content parallelization
- **Secure:** Rust language

Plan 📆

- Growing a **healthy ecosystem**
- Multiple organizations **joining efforts**
- Public and private sector **funding**

Opportunities 🌈

- **Simple applications** (controlled environment)
- **UI frameworks**
- **Default web engine** for Rust apps

The topic for today

GOSIM®
GLOBAL OPEN-SOURCE INNOVATION MEETUP

servo

The embeddable, independent, memory-safe, modular, parallel web browser engine

Parallelism vs Concurrency(logical implication)

GOSIM®
GLOBAL OPEN-SOURCE INNOVATION MEETUP

Parallelism **implies** concurrency(if parallel, then concurrent).

Code runs on:	Code uses:	**Parallel**	**Concurrent**	**Logical Result**
Multi-core	Mutli-thread	True	True	True
Single-core	Multi-thread	False	True	?
Single-core	Single-thread	False	False	?
Multi-core	Single-thread	True	False	?

Parallelism vs Concurrency(logical implication)

Parallelism **implies** concurrency(if parallel, then concurrent).

		Parallel	**Concurrent**	**Logical Result**
Multi-core	Mutli-thread	True	True	True
Single-core	Multi-thread	False	True	True
Single-core	Single-thread	False	False	True
Multi-core	Single-thread	True	False	False

Why **false => true**, and why **false => false** both evaluate to true?
Example:
if n is greater than 3, then it should be greater than 1
n > 3 should imply n > 1
Try 4, 2, and 0 for n (2 > 3 is false, but 2 > 1 is true)
On the other hand, true => false makes no sense.

Propositional logic = key to using formal methods

Parallelism(and modularity) in Servo

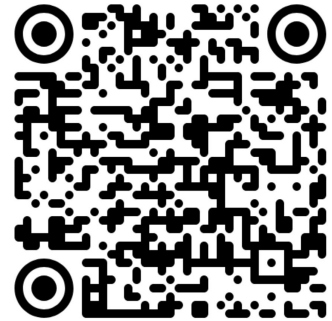
GOSIM®
GLOBAL OPEN-SOURCE INNOVATION MEETUP

Servo: large system consisting of multiple modular components.

Parallelism within a single component	Parallelization of algorithms: - Layout - Network(parallel requests) - Image decoding(parallel image decoding) - Data and Task parallelism - Usually via a thread-pool owned by component
Parallelism system-wide(in-between components)	Components usually run in their own thread, at least(sometimes in their own process too): - Message-passing is the default approach. - Task parallelism

In both cases, concurrency challenges the correctness of algorithms.
But a majority of algorithms are “simple”
Message-passing is a good default for their implementation.

Previous talk on Servo’s modularity:



Message-passing in Servo

thread + loop on receiving on channel(s) = event loop

data owned by thread(no shared data except channels).

The HTML spec,
which Servo implements,
follows a similar approach

Message-passing in HTML

GOSIM®
GLOBAL OPEN-SOURCE INNOVATION MEETUP

§ **8.1.7.3 Processing model**

An event loop must continually run through the following steps for as long as it exists:

1. Let *oldestTask* and *taskStartTime* be null.
2. If the event loop has a task queue with at least one runnable task, then:
    1. Let *taskQueue* be one such task queue, chosen in an implementation-defined manner.

    Note

    Remember that the microtask queue is not a task queue, so it will not be chosen in this microtask task source is associated might be chosen in this step. In that case, the task microtask, but it got moved as part of spinning the event loop.

    2. Set *taskStartTime* to the unsafe shared current time.
    3. Set *oldestTask* to the first runnable task in *taskQueue*, and remove it from *taskQueue*.
    4. If *oldestTask*'s document is not null, then record task start time given *taskStartTime* and
    5. Set the event loop's currently running task to *oldestTask*.
    6. Perform *oldestTask*'s steps.

Welcome to the WHATWG community

Maintaining and evolving HTML since 2004

The HTML event-loop is specified at:

https://html.spec.whatwg.org/multipage/#event-loop-processing-model

Parallelism in HTML

A **parallel queue** represents a queue of algorithm steps that must be run in series.

A parallel queue has an **algorithm queue** (a queue), initially empty.

To **enqueue steps** to a parallel queue, enqueue the algorithm steps to the parallel queue's algorithm queue.

To **start a new parallel queue**, run the following steps:

1. Let *parallelQueue* be a new parallel queue.
2. Run the following steps in parallel:
    1. While true:
        1. Let *steps* be the result of dequeueing from *parallelQueue*'s algorithm queue.
        2. If *steps* is not nothing, then run *steps*.

"*To run steps in parallel means those steps are to be run, one after another, at the same time as other logic in the standard (e.g., at the same time as the event loop).*"

Running steps in parallel is specified at:

https://html.spec.whatwg.org/multipage/#in-parallel

Steps running "in-parallel" communicate with steps running "on the event-loop" via message-passing(queuing of tasks), and vice-versa(queuing of parallel steps). Again no shared data.

Message-passing in Servo

GOSIM®
GLOBAL OPEN-SOURCE INNOVATION MEETUP

Most of the time, it works well, but…

Simple	**Complicated**
Step 1: receive message	Step 1: receive message
Step 2: handle message: apply to current state, resulting in final state	Step 2: handle message: apply to current state, resulting in an intermediate state.
Step 3: GOTO step 1, algorithm done.	Step 3: GOTO step 1, and when receiving further relevant messages, apply those to intermediate state.
(each message spawns a new instance of a simple algo)	Step 4: If in final state, GOTO step 1, algorithm done
	(complicated algo consist of multiple messages)

Complicated algorithms are challenging to describe in English: do not pay too much attention to the above. **We need more formal methods.**

Message-passing in Servo

What problems with complicated algorithms?

Intermediate state and need to “wait” for subsequent messages(but without blocking, and while continuing to handle other messages for other interleaved algorithms) can results in:

- mis-handling of state changes(sometimes caused by other interleaved algos)
- timeouts(livelocks due to never receiving the “subsequent message”)

Example of “complicated”: update the rendering

The “update the rendering” algorithm

https://html.spec.whatwg.org/multipage/#update-the-rendering

- Part of the HTML spec
- Queues a task on the HTML event-loop
- Task runs a series of (ordered) steps

A window event loop *eventLoop* must also run the following in parallel, as long as it exists:

1. Wait until at least one navigable whose active document's relevant agent's event loop is *eventLoop* might have a rendering opportunity.
2. Set *eventLoop*'s last render opportunity time to the unsafe shared current time.
3. For each *navigable* that has a rendering opportunity, queue a global task on the rendering task source given *navigable*'s active window to **update the rendering**:

The “update the rendering” algorithm

GOSIM®
GLOBAL OPEN-SOURCE INNOVATION MEETUP

Task runs long list of steps

6. For each *doc* of *docs*, reveal *doc*.
7. For each *doc* of *docs*, flush autofocus candidates for *doc* if its node navigable is a top-level traversable.
8. For each *doc* of *docs*, run the resize steps for *doc*. [CSSOMVIEW]
9. For each *doc* of *docs*, run the scroll steps for *doc*. [CSSOMVIEW]
10. For each *doc* of *docs*, evaluate media queries and report changes for *doc*. [CSSOMVIEW]
11. For each *doc* of *docs*, update animations and send events for *doc*, passing in relative high resolution time given *frameTi*
    doc's relevant global object as the timestamp [WEBANIMATIONS]
12. For each *doc* of *docs*, run the fullscreen steps for *doc*. [FULLSCREEN]

Many of the steps call into other specs(CSS, fullscreen, etc…): **hook for integration with the HTML event-loop**.

Step 14 calls into the animation frame callbacks

14. For each *doc* of *docs*, run the animation frame callbacks for *doc*, passing in the relative high resolution time given *frameTimestamp* and *doc*'s relevant global object as the timestamp.

The “update the rendering” algorithm

Servo implemented the concept earlier in 2024.

This turned many “simple” algorithms into “complicated” ones.

Example: animation frame callbacks.

Before(simple)	After(complicated)
Step 1: Whenever a relevant message is received: call into any registered animation frame callbacks	Step 1: Whenever a relevant message is received: update state(intermediate), and queue a “update the rendering” task.
Step 2: GOTO step 1	Step 2: **When task runs**: call into any registered animation frame callbacks
	Step 3: GOTO step 1

Complicated: “when task runs…”

The “update the rendering” algorithm

Problem: “when task runs…”

A note in the Spec says it all:

3. For each *navigable* that has a rendering opportunity, queue a global task on the rendering task source given *navigable*'s active window to **update the rendering**:

Note

This might cause redundant calls to update the rendering. However, these calls would have no observable effect because there will be no rendering necessary, as per the Unnecessary rendering *step. Implementations can introduce further optimizations such as only queuing this task when it is not already queued. However, note that the document associated with the task might become inactive before the task is processed.*

The “update the rendering” algorithm

GOSIM®
GLOBAL OPEN-SOURCE INNOVATION MEETUP

Problem: “when task runs…”

Note: an HTML event-loop can run more than one document(mostly same-origin iframes or tabs). Documents are also knowns as Pipelines in Servo speak.

Servo’s initial buggy implementation:

- Use a single boolean to batch “update the rendering” for all documents in an HTML event-loop.
- Task is still linked to a particular document(a.k.a Pipeline).
- If the document becomes inactive before the task runs: livelock

The “update the rendering” algorithm

GOSIM®
GLOBAL OPEN-SOURCE INNOVATION MEETUP

The fix: switch from a boolean to a hashset. Only “batching” the task per pipeline.

```
520     -   has_queued_update_the_rendering_task: DomRefCell<bool>,
    520 +   #[no_trace]
    521 +   update_the_rendering_task_queued_for_pipeline: DomRefCell<HashSet<PipelineId>>,
```

Note: code is simple, but algorithm behind the code is not.

Also, problems only surface from time to time(intermittent), and usually not when you just run tests on CI or locally for a single PR. Only a day or two later after a lot of other CI runs from unrelated PRs…

If testing doesn’t help catch the error in a timely manner, what can?

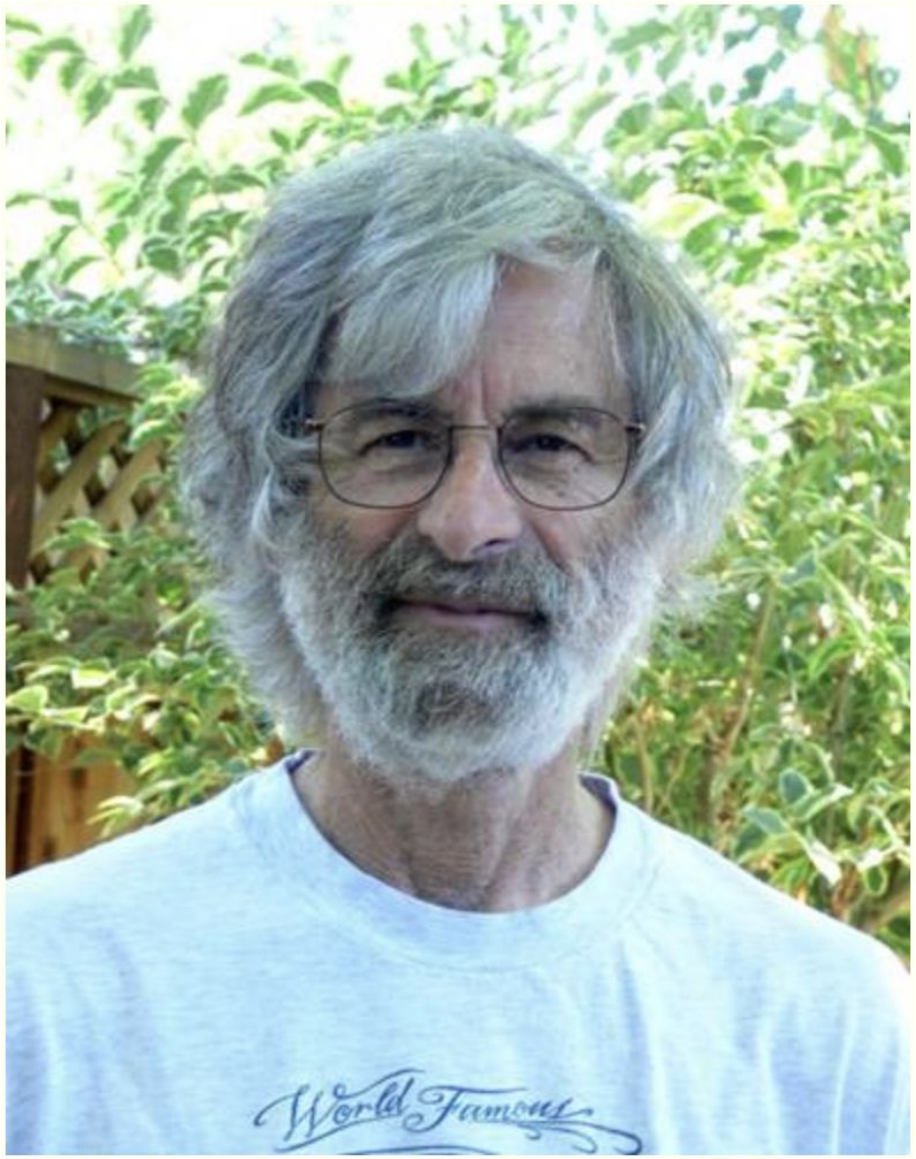
Enter: TLA+

What is TLA+?

TLA+ is a high-level language for modeling programs and systems--especially concurrent and distributed ones. It's based on the idea that the best way to describe things precisely is with simple mathematics. TLA+ and its tools are **useful for eliminating fundamental design errors**, which are hard to find and expensive to correct in code.

Created by Leslie Lamport (Turing award 2013) at Microsoft.

https://lamport.azurewebsites.net/tla/tla.html



Use TLA+ to fix Servo in five easy steps

GOSIM®
GLOBAL OPEN-SOURCE INNOVATION MEETUP

1. There is a concurrent problem(intermittent test failure).
2. You have an idea what the problem could be.
3. Model the failure.
4. Model the fix.
5. Fix the code.

Ideally: model the system before writing code…

Use TLA+ to fix Servo: notice problem

GOSIM®
GLOBAL OPEN-SOURCE INNOVATION MEETUP

WPT Test fails intermittently.

WPT = [web-platform-tests]

A test suite for the Web stack shared across browser engines, including Servo.

cbc9304 servo / tests / wpt / tests / html / webappapis / update-rendering / child-document-raf-order.html

mrobinson Move tests/wpt/web-platform-tests to tests/wpt/tests

Code Blame 118 lines (90 loc) · 3.94 KB

```
1   <!DOCTYPE HTML>
2   <meta charset=UTF-8>
3   <title>Ordering of steps in "Update the Rendering" - child document requestAnimationFrame order</title>
4   <link rel="help" href="https://html.spec.whatwg.org/multipage/webappapis.html#update-the-rendering">
5   <link rel="author" title="L. David Baron" href="https://dbaron.org/">
6   <link rel="author" title="Mozilla" href="https://mozilla.org/">
7   <script src="/resources/testharness.js"></script>
8   <script src="/resources/testharnessreport.js"></script>
9
10  <div id=log></div>
11
12  <!--
13
14  This test tests the interaction of just two substeps of the "Update the
15  rendering" steps in
16  https://html.spec.whatwg.org/multipage/webappapis.html#update-the-rendering
```


Use TLA+ to fix Servo: theorize problem

GOSIM®
GLOBAL OPEN-SOURCE INNOVATION MEETUP

You have an idea what the problem could be.

3. For each *navigable* that has a rendering opportunity, queue a global task on the rendering task source given *navigable*'s active window to **update the rendering**:

Note

This might cause redundant calls to update the rendering. However, these calls would have no observable effect because there will be no rendering necessary, as per the Unnecessary rendering *step. Implementations can introduce further optimizations such as only queuing this task when it is not already queued. However, note that the document associated with the task might become inactive before the task is processed.*

Use TLA+ to fix Servo: model failure

Finally, some TLA+

Let’s go through a spec.

RenderingUpdateBatchingBroken.tla

```
---------------------- MODULE RenderingUpdateBatchingBroken ----------------------
EXTENDS FiniteSets, Naturals
VARIABLES task_queue, rendering_task_queued
CONSTANT N
------------------------------------------------------------------------------
Pipeline == 0..N

TypeOk == /\ task_queue \in [Pipeline -> BOOLEAN]
          /\ rendering_task_queued \in BOOLEAN

QueuedTaskRuns == rendering_task_queued => \E p \in Pipeline: task_queue[p]
------------------------------------------------------------------------------

Init == /\ task_queue = [p \in Pipeline |-> FALSE]
        /\ rendering_task_queued = FALSE

QueueTask(p) == /\ rendering_task_queued = FALSE
                /\ rendering_task_queued' = TRUE
                /\ task_queue' = [task_queue EXCEPT ![p] = TRUE]

RunTask(p) == /\ task_queue[p] = TRUE
              /\ task_queue' = [task_queue EXCEPT ![p] = FALSE]
              /\ rendering_task_queued' = FALSE

ClosePipeline(p) == /\ task_queue' = [task_queue EXCEPT ![p] = FALSE]
                    /\ UNCHANGED<<rendering_task_queued>>

Next == \E p \in Pipeline: \/ QueueTask(p)
                           \/ RunTask(p)
                           \/ ClosePipeline(p)

Spec  ==  Init /\ [][Next]_<<task_queue, rendering_task_queued>>
------------------------------------------------------------------------------
THEOREM  Spec  =>  [](TypeOk /\ QueuedTaskRuns)
==============================================================================
```

Intro to a TLA+ spec

Let’s use Rust programming talk for easy intro to somewhat similar concepts.

RenderingUpdateBatchingBroken.tla

```
1  ---------------------- MODULE RenderingUpdateBatchingBroken ----------------------
2  EXTENDS FiniteSets, Naturals
3  VARIABLES task_queue, rendering_task_queued
4  CONSTANT N
5  ------------------------------------------------------------------------------
6  Pipeline == 0..N
7
```

MODULE = mod
EXTENDS = use
VARIABLES = let
CONSTANT = declare config value.

PIPELINE = const PIPELINES: HashSet<PipelineId> = // init with config.

Intro to a TLA+ spec

Init =
    let mut spec = Spec {task_queue: ..,
            rendering_task_queued: ..};

[p \in Pipeline |-> FALSE] is like
let mut task_queue:
HashMap<PipelineId, bool> =
Default::default(); // Add loop to init
all keys to false.

QueueTask, RunTask,
and ClosePipeline are all actions
(like methods).

Primed variables like
task_queue' mean “mutation”

The other mean “condition” on the action

```
12  ----------------------------------------------------------------------------
13
14  Init == /\ task_queue = [p \in Pipeline |-> FALSE]
15          /\ rendering_task_queued = FALSE
16
17  QueueTask(p) == /\ rendering_task_queued = FALSE
18                  /\ rendering_task_queued' = TRUE
19                  /\ task_queue' = [task_queue EXCEPT ![p] = TRUE]
20
21  RunTask(p) == /\ task_queue[p] = TRUE
22                /\ task_queue' = [task_queue EXCEPT ![p] = FALSE]
23                /\ rendering_task_queued' = FALSE
24
25  ClosePipeline(p) == /\ task_queue' = [task_queue EXCEPT ![p] = FALSE]
26                      /\ UNCHANGED<<rendering_task_queued>>
27
```

/\ means “and”

But remember: TLA+ is nothing like programming.

Intro to a TLA+ spec

```
28   Next == \E p \in Pipeline: \/ QueueTask(p)
29                              \/ RunTask(p)
30                              \/ ClosePipeline(p)
31
32   Spec  ==  Init /\ [][Next]_<<task_queue, rendering_task_queued>>
33   ----------------------------------------------------------------------
```

\/ means “or”

Next is like: select! {} // select an action ready to run.

Spec is like:
// Run Init.
let mut spec = init();
loop {
    next(&mut spec);
 }

[] means “always true”

\E p \in Pipeline: means “for one pipeline in the set of pipeline”

Intro to a TLA+ spec

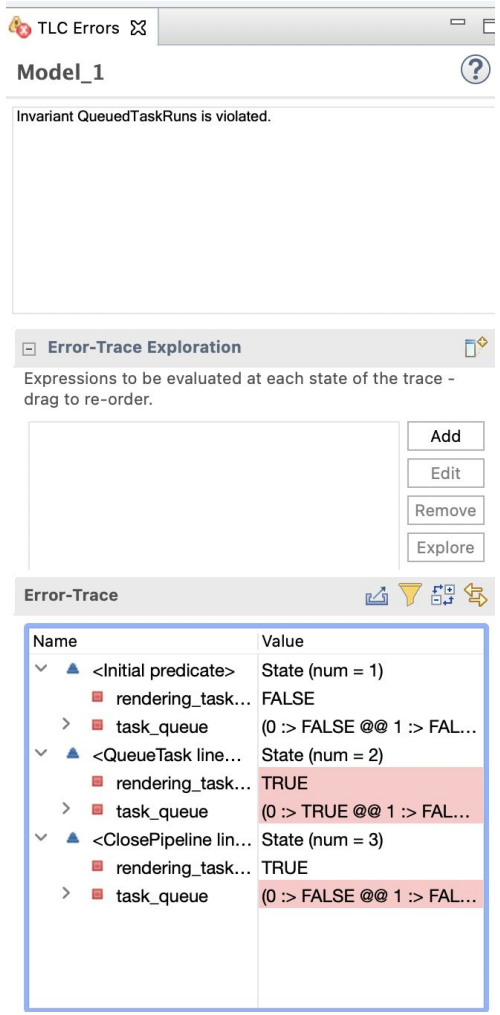
```
33  ----------------------------------------------------------------------------
34  THEOREM  Spec  =>  [](TypeOk /\ QueuedTaskRuns)
35  ============================================================================
```

=> means “implies”. So If Spec is True, then TypeOk and QueuedTaskRuns must be True as well.

```
 7
 8  TypeOk == /\ task_queue \in [Pipeline -> BOOLEAN]
 9            /\ rendering_task_queued \in BOOLEAN
10
11  QueuedTaskRuns == rendering_task_queued => \E p \in Pipeline: task_queue[p]
12  ----------------------------------------------------------------------------
```

TypeOk and QueuedTaskRuns are the invariants of the spec.

Use TLA+ to fix Servo: model failure



The TLC model checker is very good at catching errors.

Use TLA+ to fix Servo: model failure

State 1: Init

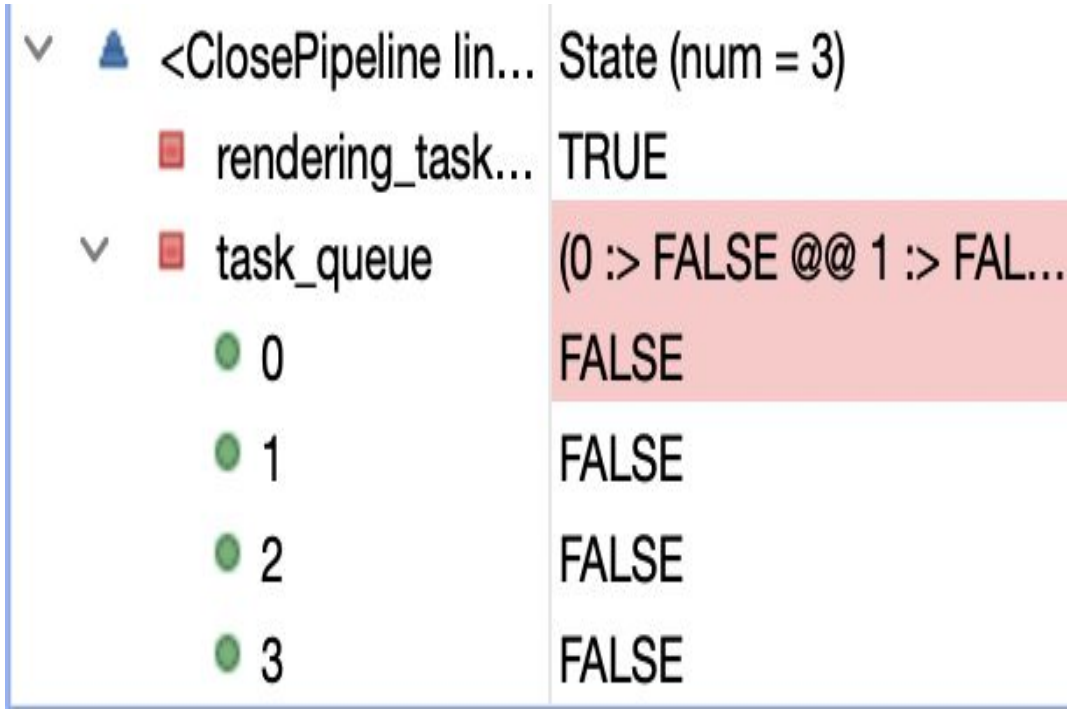
<Initial predicate>	State (num = 1)
rendering_task…	FALSE
task_queue	(0 :> FALSE @@ 1 :> FAL…
0	FALSE
1	FALSE
2	FALSE
3	FALSE

Use TLA+ to fix Servo: model failure

State 2: QueueTask

<QueueTask line…	State (num = 2)
rendering_task…	TRUE
task_queue	(0 :> TRUE @@ 1 :> FAL…
0	TRUE
1	FALSE
2	FALSE
3	FALSE

Use TLA+ to fix Servo: model failure



<ClosePipeline lin...	State (num = 3)
rendering_task...	TRUE
task_queue	(0 :> FALSE @@ 1 :> FAL...
0	FALSE
1	FALSE
2	FALSE
3	FALSE

State 3: ClosePipeline

```
rendering_task_queued => \E p \in Pipeline: task_queue[p]
```

rendering_task_queued is **TRUE**

\E p \in Pipeline: task_queue[p] is **FALSE**

Remember logical implication?
True and False = False

Invariant is not maintained.

(In-)Formal methods in Servo

GOSIM®
GLOBAL OPEN-SOURCE INNOVATION MEETUP

Intermezzo: why the (in-)? Because we're not being very formal.

- No proof of correctness for the spec(see TLAPS proof system).
- No verification of the code(I have no idea how to do that).
- We don't even used advanced TLA+(like "eventually happens").

Rather, we:

- Use the TLC model checker to gain confidence on the correctness of our spec(never 100%)
- Use our brain to gain confidence that the code implements the spec.
- logical unit-test: absence of bugs not proven.
- Lowest possible effort
- Writing a spec takes 30-60 min, but usually takes two iteration over two days(a night of sleep) to figure out some things.

Comparison with default code-only approach: debugging concurrency takes days, weeks, and sometimes "forever"(ignore problem)

More info at:
https://lamport.azurewebsites.net/pubs/toolbox.pdf

Use TLA+ to fix Servo: model the fix

GOSIM®
GLOBAL OPEN-SOURCE INNOVATION MEETUP

Let’s write a spec for the fix: our new init(and new variable).

```
17      Init == /\ task_queue = [p \in Pipeline |-> FALSE]
    -           /\ rendering_task_queued = FALSE
18  +           /\ rendering_task_queued = [p \in Pipeline |-> FALSE]
19  +           /\ closed = {}
```

Note: {} is like HashSet::new();

Use TLA+ to fix Servo: model fix

Let's write a spec for the fix:
our new actions.

```
    - QueueTask(p) == /\ rendering_task_queued = FALSE
    -                 /\ rendering_task_queued' = TRUE
21  + QueueTask(p) == /\ p \notin closed
22  +                 /\ rendering_task_queued[p] = FALSE
23  +                 /\ rendering_task_queued'
24  +                     = [rendering_task_queued EXCEPT ![p] = TRUE]
25                    /\ task_queue' = [task_queue EXCEPT ![p] = TRUE]
26  +                 /\ UNCHANGED<<closed>>
27
28    RunTask(p) == /\ task_queue[p] = TRUE
29                  /\ task_queue' = [task_queue EXCEPT ![p] = FALSE]
    -               /\ rendering_task_queued' = FALSE
30  +               \* The below resets the batching for all pipelines.
31  +               /\ rendering_task_queued' = [pp \in Pipeline |-> FALSE]
32  +               /\ UNCHANGED<<closed>>
33
34    ClosePipeline(p) == /\ task_queue' = [task_queue EXCEPT ![p] = FALSE]
35  +                     /\ closed' = closed \cup {p}
36                        /\ UNCHANGED<<rendering_task_queued>>
```

\cup is set union

Use TLA+ to fix Servo: model fix

```
 9     TypeOk == /\ task_queue \in [Pipeline -> BOOLEAN]
-              /\ rendering_task_queued \in BOOLEAN
10    +        /\ rendering_task_queued \in [Pipeline -> BOOLEAN]
11    +        /\ closed \in SUBSET Pipeline
12
-     - QueuedTaskRuns == rendering_task_queued => \E p \in Pipeline: task_queue[p]
13    + QueuedTaskRuns == \A p \in Pipeline \ closed:
14    +                     rendering_task_queued[p] => task_queue[p]
```

Let's write a spec for the fix:
our new invariants.

\ is set difference

\A p \in Pipeline \ closed means: "for all pipelines that are not closed"

Logical implication: rendering_task_queued can be False while task_queue is True: this is what happens when batching is reset for all pipelines while there are tasks pending.

Intermezzo: English vs Math

GOSIM®
GLOBAL OPEN-SOURCE INNOVATION MEETUP

Defining invariants: English vs TLA+

Note

This might cause redundant calls to update the rendering. However, these calls would have no observable effect because there will be no rendering necessary, as per the Unnecessary rendering *step. Implementations can introduce further optimizations such as only queuing this task when it is not already queued. However, note that the document associated with the task might become inactive before the task is processed.*

English: verbose, vague, and validity cannot be computed.

```
QueuedTaskRuns == \A p \in Pipeline \ closed:
                    rendering_task_queued[p] => task_queue[p]
```

TLA+: short, precise, and TLC can check it.

Use TLA+ to fix Servo: model fix

GOSIM®
GLOBAL OPEN-SOURCE INNOVATION MEETUP

Model Checking Results

Let's run TLC again

General

Start: 18:16:28 (Oct 12) **End: 18:16:35 (Oct 12)**

Fingerprint collision probability: calculated: 1.3E-13

Statistics

State space progress (click column header for graph)

Time	Diameter	States Found	Distinct States	Queue Size
00:00:07	10	4,489	624	0
00:00:06	0	1	1	1

7 seconds, 4k+ states explored…

Use TLA+ to fix Servo: fix the code

GOSIM®
GLOBAL OPEN-SOURCE INNOVATION MEETUP

```
520       -     has_queued_update_the_rendering_task: DomRefCell<bool>,
      520 +     #[no_trace]
      521 +     update_the_rendering_task_queued_for_pipeline: DomRefCell<HashSet<PipelineId>>,
```

The easy part.

My favorite TLA+ resource

https://cseweb.ucsd.edu/classes/sp05/cse128/

A class from 2005 by Keith Marzullo
at the University of San Diego.

The end. Thank you for listening.

My github: https://github.com/gterzian

Previous talk on Servo's modularity:

GOSIM®
GLOBAL OPEN-SOURCE INNOVATION MEETUP

https://servo.org

https://floss.social/@servo

https://twitter.com/ServoDev

https://book.servo.org/

servo

THE LINUX FOUNDATION | Europe

GOSIM CHINA 2024

GOSIM®
GLOBAL OPEN-SOURCE INNOVATION MEETUP

GOSIM CHINA 2024

THANK YOU