TLA+ Mutex Lock Specification

CS254 Final Project

March 24, 2025

1 Overview

This document explains the TLA+ specification of a mutex lock system with FIFO queuing. The specification ensures both safety (mutual exclusion) and liveness (no starvation) properties.

2 State Space

The specification uses three state variables:

- `pc`: Program counter for each process (noncritical, trying, critical)
- `lock`: Lock state (0 for free, or process ID that holds it)
- `queue`: Sequence of processes waiting for the lock

3 State Diagram

$$\begin{tikzcd} \text{noncritical} \arrow[r, "\text{Try}"] & \text{trying} \arrow[r, "\text{Enter}"] & \text{critical} \arrow[ll, bend right, "\text{Exit}"] \end{tikzcd}$$

4 Actions

4.1 Try

When a process wants to enter the critical section:

- Changes state from noncritical to trying
- Appends itself to end of queue

4.2 Enter

A process can enter the critical section when:

- It is in trying state
- The lock is free (lock = 0)
- It is at the head of the queue

4.3 Exit

When a process is done:

- Releases the lock
- Returns to noncritical state

5 Properties

5.1 Safety

Mutual Exclusion: No two processes can be in the critical section simultaneously.

```
MutualExclusion ==
    \A p1, p2 \in Procs :
        (p1 # p2) => ~(pc[p1] = "critical" /\ pc[p2] = "critical")
```

5.2 Liveness

No Starvation: If a process is in the queue, it eventually enters the critical section.

```
NoStarvation ==
    \A p \in Procs : InQueue(p) ~> (pc[p] = "critical")
```

6 FIFO Queue Implementation

The specification uses a sequence to implement strict FIFO ordering:

- New processes are added to end: `Append(queue, p)`
- Only head of queue can enter: `Head(queue) = p`
- Process removed from front when entering: `Tail(queue)`

This ensures processes enter the critical section in exactly the order they requested access.

7 Fairness

The specification includes weak fairness conditions for all actions:

```
Fairness == \A p \in Procs :
    /\ WF_vars(Try(p))
    /\ WF_vars(Enter(p))
    /\ WF_vars(Exit(p))
```

This ensures that if an action is continuously enabled, it will eventually be taken.