A Simple Critical Section Protocol

There are \( N \) concurrent processes \( P_1, \ldots, P_N \) that share some data. A process accessing the shared data is said to execute in its critical section.

\( P_i \) executes the following code:

```
repeat for ever:
  external;
  critical section(cs);
```

For simplicity, we assume the external section is one instruction.
**Mutual exclusion requirement**: at most one process executing in its critical section at a given moment in time.

Idea: use a **semaphore** to coordinate the processes.

- the semaphore ($Sem$) can hold 2 values: 0(red) and 1(green)
- initially $Sem$ is 1
- the semaphore access operations are *request* and *release*
•when a process $P_i$ wants to execute its $cs$, it requires access to the semaphore

  •if $Sem$ is 1 then $P_i$ starts executing the $cs$; $Sem$ becomes 0

  •if $Sem$ is 0 then $P_i$ waits

•when $P_i$ finishes executing the $cs$, it releases the semaphore; $Sem$ becomes 1 and some waiting process may start executing its $cs$.

More realistic semaphores (with integer values and waiting queues) will be considered later.
Each process $P_i$ is executing this simple mutual-exclusion protocol with one semaphore:

```
repeat for ever:

0 external;
1 request;
2 critical section(cs);
3 release;
```
The goal of the paper is to prove the protocol guarantees mutual exclusion.

There are multiple ways to prove the protocol correct. Two possible approaches:

1. using global states and histories
2. using events described by means of predicates and functions.

The paper advocates using the 2\textsuperscript{nd} approach to proving concurrency protocols.
First Proof: Global States and Histories
(the classical approach)

For each process $P_i$ the program counter variable $PC_i$ is considered and $type(PC_i) = \{0, 1, 2, 3\}$.

Also $type(Sem) = \{0, 1\}$.

Thus we have $N+1$ variables in the system:

$$V = \{PC_1, \ldots, PC_N, Sem\}.$$

A global state is a function $S$ with domain $V$ such that each variable takes values in its corresponding type; $S$ describes a possible instantaneous situation of all processes and the semaphore.
To model a possible history of the system we repeatedly apply one of the following steps:

1. **Executions of line 0 by** \(P_i\): from \(S_1\) with \(S_1(PC_i)=0\) to \(S_2\) that differs from \(S_1\) only by \(S_2(PC_i)=1\)

2. **Executions of line 1 by** \(P_i\): from \(S_1\) with \(S_1(PC_i)=1\) and \(S_1(Sem)=1\) to \(S_2\) that differs from \(S_1\) only by \(S_2(PC_i)=2\) and \(S_2(Sem)=0\)

3. **Executions of line 2 by** \(P_i\): from \(S_1\) with \(S_1(PC_i)=2\) to either \(S_2=S_1\) or \(S_2\) that differs from \(S_1\) only by \(S_2(PC_i)=3\)

4. **Executions of line 3 by** \(P_i\): from \(S_1\) with \(S_1(PC_i)=3\) and \(S_1(Sem)=0\) to \(S_2\) that differs from \(S_1\) only by \(S_2(PC_i)=0\) and \(S_2(Sem)=1\).
The initial state $S_0$ has $S_0(PC_i) = 0$ for all $i$ and $S_0(Sem) = 1$.

Then the mutual exclusion condition is written as:

If $<S_i| i>$ is a history, and if $k$ is any index, then it is not the case that $S_k(PC_i) = 2$ and $S_k(PC_j) = 2$ for different $i$ and $j$.

This can be shown with the aid of the following lemma proven by induction on $k$:

$\forall 1 \leq i \leq N : S_k(PC_i) \in \{2,3\} \rightarrow [\forall 1 \leq j \leq N (j \neq i \rightarrow S_k(PC_j) \in \{0,1\}) \land S_k(Sem) = 0]$