MODULE *peterson_lock*

Peterson's Lock was first formalised in

G.L. Peterson : *Myths* About the Mutual Exclusion Problem

Peterson's algorithm is a concurrent algorithm for solving the problem of mutual exclusion that allows multiple processes to sgare a single resource without conflicts Several versions of the lock exist. This specification formally presents the version that works with 2 processes.

$\textsc{extends}\ Integers,\ TLAPS$

$\textsc{variables}\ turn,\ state,\ flag$

$vars \triangleq \langle turn,\ state,\ flag\rangle$

$ProcSet \triangleq \{0,\ 1\}$

$States \triangleq \{\text{“Start”},\ \text{“RequestTurn”},\ \text{“Waiting”},\ \text{“CriticalSection”}\}$

$Not(i) \triangleq \textsc{if}\ i = 0\ \textsc{then}\ 1\ \textsc{else}\ 0$

State *Initialisation* for the processes. At the start of the logical time, both the processes have their flags set to false and the turn can be 0 or 1. The initial process control points to 'Start' for all the processes

$$
\begin{aligned}
Init \triangleq\ & \land flag = [i \in ProcSet \mapsto \textsc{false}] \\
& \land turn \in \{0,\ 1\} \\
& \land state = [i \in ProcSet \mapsto \text{“Start”}]
\end{aligned}
$$

When the process makes progress, the first step is to capture the flag. The process does this by setting its entry in the list of flags to TRUE. The process can then request its turn.

$$
\begin{aligned}
& SetFlag(p) \triangleq \\
& \qquad \land state[p] = \text{“Start”} \\
& \qquad \land flag' = [flag\ \textsc{except}\ ![p] = \textsc{true}] \\
& \qquad \land state' = [state\ \textsc{except}\ ![p] = \text{“RequestTurn”}] \\
& \qquad \land \textsc{unchanged}\ \langle turn\rangle
\end{aligned}
$$

After capturing the flag, the process indicates its intent by setting the turn to The processes then contend to get their turn. Process $P1$ gives the turn to $P2$ and vice a versa. The process that comes first and has turn set to its entry then proceeds into the critical section. The other process meanwhile enters the waiting state.

$$
\begin{aligned}
& SetTurn(p) \triangleq \\
& \qquad \land state[p] = \text{“RequestTurn”} \\
& \qquad \land turn' = 1 - p \\
& \qquad \land state' = [state\ \textsc{except}\ ![p] = \text{“Waiting”}] \\
& \qquad \land \textsc{unchanged}\ \langle flag\rangle
\end{aligned}
$$

In the waiting state, the process keeps checking for its turn. If it observes the turn set to its own process identifier or notices that the other process has given up the intent to move into the critical section by setting its flag to false, the waiting process enters the critical section.

$$
\begin{aligned}
&\mathit{EnterCriticalSection}(p) \triangleq \\
&\qquad \land\ \mathit{state}[p] = \text{“Waiting”} \\
&\qquad \land\ (\mathit{flag}[(1-p)] = \text{FALSE} \lor \mathit{turn} = p) \\
&\qquad \land\ \mathit{state}' = [\mathit{state}\ \text{EXCEPT}\ ![p] = \text{“CriticalSection”}] \\
&\qquad \land\ \text{UNCHANGED}\ \langle \mathit{turn}, \mathit{flag} \rangle
\end{aligned}
$$

After exiting the critical section, cleanup is performed where the process sets its flag to false. This lets other processes proceed.

$$
\begin{aligned}
&\mathit{ExitCriticalSection}(p) \triangleq \\
&\qquad \land\ \mathit{state}[p] = \text{“CriticalSection”} \\
&\qquad \land\ \mathit{flag}' = [\mathit{flag}\ \text{EXCEPT}\ ![p] = \text{FALSE}] \\
&\qquad \land\ \mathit{state}' = [\mathit{state}\ \text{EXCEPT}\ ![p] = \text{“Start”}] \\
&\qquad \land\ \text{UNCHANGED}\ \langle \mathit{turn} \rangle
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{Next} \triangleq \exists\, p \in \mathit{ProcSet} : \\
&\qquad \lor\ \mathit{SetFlag}(p) \\
&\qquad \lor\ \mathit{SetTurn}(p) \\
&\qquad \lor\ \mathit{EnterCriticalSection}(p) \\
&\qquad \lor\ \mathit{ExitCriticalSection}(p)
\end{aligned}
$$

$$\mathit{Spec} \triangleq \mathit{Init} \land \Box[\mathit{Next}]_{\mathit{vars}}$$

$$
\begin{aligned}
&\mathit{proc}(\mathit{self}) \triangleq \\
&\qquad \lor\ \mathit{SetFlag}(\mathit{self}) \\
&\qquad \lor\ \mathit{SetTurn}(\mathit{self}) \\
&\qquad \lor\ \mathit{EnterCriticalSection}(\mathit{self}) \\
&\qquad \lor\ \mathit{ExitCriticalSection}(\mathit{self})
\end{aligned}
$$

$$\mathit{SpecWithFairness} \triangleq \mathit{Spec} \land \mathrm{WF}_{\mathit{vars}}(\mathit{Next}) \land \forall\, p \in \{0, 1\} : \mathrm{WF}_{\mathit{vars}}(\mathit{SetFlag}(p))$$

The execution invariant assers the following:

1. For all the processes, If the process has proceeded from the start state, it should have set its flag.
2. If a process is in the critical section, then no other process shoudl be in the critical section
3. For any process that is waiting, the turn should be set to the process that is making progress.

$ExecutionInvariant \triangleq \forall i \in ProcSet :$
$$\begin{array}{l} \land\ state[i] \in States \setminus \{\text{"Start"}\} \Rightarrow flag[i] \\ \land\ state[i] \in \{\text{"CriticalSection"}\} \Rightarrow \begin{array}{l} \land\ state[Not(i)] \notin \{\text{"CriticalSection"}\} \\ \land\ state[Not(i)] \in \{\text{"Waiting"}\} \Rightarrow turn = i \end{array} \end{array}$$

The Type invariant assers the following:

1. The state for any process shoule be one of the predefined states i.e. Start, Set Flag, Set Turn, Waiting or Critical section.
2. Turn should only be set to valid process identifiers.
3. Flag should be set to either TRUE or FALSE.

$$TypeInvariant \triangleq \begin{array}{l} \land\ state \in [\{0, 1\} \rightarrow States] \\ \land\ turn \in \{0, 1\} \\ \land\ flag \in [\{0, 1\} \rightarrow \{\text{TRUE}, \text{FALSE}\}] \end{array}$$

The mutual exclusion property ensures that no 2 processes enter the critical section at the same time. For every possible execution of the specification, in every state, the property should hold true.

$$MutualExclusion \triangleq \neg(state[0] = \text{"CriticalSection"} \land state[1] = \text{"CriticalSection"})$$

$$Inv \triangleq ExecutionInvariant \land TypeInvariant$$

Formal proof that *Petersons* algorithm solves mutual exclusion.

The proof is a standard invariance proof that assers that any step of the algorithm starting in a state in which an invariant is true leaves the invariant true.

$\langle 1\rangle 1$ Asserts that the Execution and Type Invariants are true in the Initial state. The proof for this step is straightforward.

$\langle 1\rangle 3$ Asserts that the Execution and Type Invariants together imply Mutual exclusion. This step is proved using the results from step $\langle 1\rangle 2$.

$\langle 1\rangle 2$ Asserts that if the Invariants are true and any process makes progress,the invariants still hold true in the next state.

THEOREM $Spec \Rightarrow \Box MutualExclusion$
PROOF
  $\langle 1\rangle 1.\ Init \Rightarrow Inv$
    BY DEFS $Init, Inv, TypeInvariant, ExecutionInvariant, vars, States, ProcSet$

  $\langle 1\rangle 2.\ Inv \land [Next]_{vars} \Rightarrow Inv'$
    $\langle 2\rangle 1.$ SUFFICES ASSUME Inv, $Next$PROVE Inv'
      BY DEFS $ExecutionInvariant, TypeInvariant, Inv, vars$

$\langle 2 \rangle 2.$ *TypeInvariant'*

BY $\langle 2 \rangle 1$

DEFS *Inv*, *TypeInvariant*, *Next*, *proc*, *Not*, *States*, *ProcSet*, *SetFlag*, *SetTurn*, *EnterCriticalSection*, *ExitCriticalSection*

For this part of the proof, we expand the invariants on randomly chosen processes from the process identifiers. The two cases possible are $\langle 3 \rangle 3$ and $\langle 3 \rangle 4$

$\langle 2 \rangle 3.$ *ExecutionInvariant'*

$\langle 3 \rangle 1.$ SUFFICES ASSUME NEW $j \in ProcSet$PROVE $ExecutionInvariant!(j)'$BY

DEF *ExecutionInvariant*, *ProcSet*

$\langle 3 \rangle 2.$ PICK $i \in ProcSet : proc(i)$

BY $\langle 2 \rangle 1$

DEF *Next*, *ProcSet*, *proc*, *SetFlag*, *SetTurn*, *EnterCriticalSection*, *ExitCriticalSection*

If we select the same process and check the execution invariants, the proof is obvious

$\langle 3 \rangle 3.$CASE $i = j$

BY $\langle 2 \rangle 1$, $\langle 3 \rangle 2$, $\langle 3 \rangle 3$

DEFS *ExecutionInvariant*, *TypeInvariant*, *Inv*, *proc*, *Not*, *ProcSet*, *SetFlag*, *SetTurn*, *EnterCriticalSection*, *ExitCriticalSection*

If we select different processes and check the execution invariants, The invariants are expanded and the proof is obvious.

$\langle 3 \rangle 4.$CASE $i \neq j$

BY $\langle 2 \rangle 1$, $\langle 3 \rangle 2$, $\langle 3 \rangle 3$

DEFS *ExecutionInvariant*, *TypeInvariant*, *Inv*, *proc*, *Not*, *ProcSet*, *SetFlag*, *SetTurn*, *EnterCriticalSection*, *ExitCriticalSection*

$\langle 3 \rangle$.QED BY $\langle 3 \rangle 3$, $\langle 3 \rangle 4$

The subparts of the above proof are collected in the QED step to prove Type invariance and Execution Invariance after every step taken according to the next state predicate

$\langle 2 \rangle 4.$ QED BY $\langle 2 \rangle 2$, $\langle 2 \rangle 3$ DEF *Inv*

$\langle 1 \rangle 3.$ $Inv \Rightarrow MutualExclusion$

BY DEF *Inv*, *MutualExclusion*, *ProcSet*, *Not*, *ExecutionInvariant*, *TypeInvariant*

$\langle 1 \rangle 4.$ QED BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$, $\langle 1 \rangle 3$, *PTL*

DEF *MutualExclusion*, *Spec*, *ExecutionInvariant*, *TypeInvariant*, *Inv*, *proc*, *Not*, *ProcSet*, *SetFlag*, *SetTurn*, *EnterCriticalSection*, *ExitCriticalSection*

* Modification History
* Last modified *Thu Dec* 10 00:19:14 *CET* 2020 by *pandey*
* Last modified *Wed Nov* 11 14:06:41 *CET* 2020 by *ayushpandey*
* Created *Tue Nov* 03 21:42:32 *CET* 2020 by *ayushpandey*