Verification and Model Checking

Seminar: Software Engineering in Winter term 2020

Ayush Pandey
Supervised by: Dr. rer. nat. Annette Bieniusa

Department of Computer Science
Technische Universität Kaiserslautern

January 11, 2021

Outline

1. Software Correctness
   - Why is Verification Important?
   - What is System/Software Verification
2. Target Problem: Peterson's Algorithm
3. Properties Satisfied by Peterson's Algorithm
4. Verification Techniques
5. Specifying Peterson's Algorithm in TLA^+
6. Model Checking Peterson's Algorithm
7. TLA^+Proof System Architecture
8. Proving Mutual Exclusion for Peterson's Algorithm in TLA^+
9. Conclusion

Why is Verification Important?

- Increasing complexity of software systems
- Time and effort constraints
- Independence from programming language constructs
- Catching errors early on

What is System/Software Verification

Definition

System/Software Verification aims towards checking whether the specified system fulfils the qualitative requirements that have been identified.

Verification $\neq$ Testing

Although software testing is very useful in identifying bugs in the system constructed from a given specification, Software verification allows for a more robust and extensive checking. [1]

Target Problem: Peterson's Algorithm

- Classical algorithm for mutual exclusion
- Specifies concurrency control for multiple processes
- Our focus: Peterson's algorithm for 2 processes

Peterson's Algorithm

```
bool flag :[false,false]
int turn
```

```
For process P0:

flag[0] = true;

turn = 1;

while (flag[1] == true && turn ==1)
{
      //Busy Waiting
}

//Critical Section

flag[0] = false;
```

```
For process P1:

flag[1] = true;

turn = 0;

while (flag[0] == true && turn ==0)
{
      //Busy Waiting
}

//Critical Section

flag[1] = false;
```

Properties Satisfied by Peterson's Algorithm

- **Mutual Exclusion**: P_0 and P_1 are not in the critical section at the same time
- **Progress**: Either P_0 or P_1 is always able to make progress
- **Bounded Waiting**: Neither P_0 nor P_1 has to wait indefinitely before entering the critical section[1]

Thought !!

What could be a possible way to ensure that these properties hold?

[1] This condition is subject to constraints specified by the scheduling method and the process priorities. For our concern, both processes have the same scheduling priority and the process scheduler does not favour one process over the other.

Verification Techniques

Model Checking

For a specification, *systematically* check a clause P on all states.
Applicable if the system generates a (finite) behavioural model.

Deduction

For a specification, provide a formal *proof* that a clause P holds.
Applicable if the system follows a mathematical theory.

MODULE **Module name**

Imports
Variables
Initial predicate
State predicates
Next relation
Property definitions

Remark

Every specification has at least one Initial predicate, and a Next relation.

MODULE *peterson_lock*

EXTENDS *Integers*, *TLAPS*

VARIABLES *turn*, *state*, *flag*

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

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

$States \triangleq \{"Start", "RequestTurn", "Waiting", "CriticalSection"\}$

$Not(i) \triangleq 1 - i$

Remark

Not(i) is a mathematical function. Such functions are called Operators in TLA$^+$.

Specifying Peterson's Algorithm in TLA$^+$

Specification State Predicates

MODULE *peterson_lock*

.....contd.

$$Init \triangleq flag = [i \in ProcSet \mapsto \text{FALSE}]$$
$$\land\ Turn \in \{0, 1\}$$
$$\land\ state = [i \in ProcSet \mapsto \text{"}Start\text{"}]$$

$$SetFlag(p) \triangleq state[p] = \text{"}Start\text{"}$$
$$\land\ flag' = [flag \text{ EXCEPT } ![p] = \text{TRUE}]$$
$$\land\ state' = [state \text{ EXCEPT } ![p] = \text{"}RequestTurn\text{"}]$$
$$\land\ \text{UNCHANGED } \langle turn \rangle$$

$$SetTurn(p) \triangleq state[p] = \text{"}RequestTurn\text{"}$$
$$\land\ turn' = Not(p)$$
$$\land\ state' = [state \text{ EXCEPT } ![p] = \text{"}Waiting\text{"}]$$
$$\land\ \text{UNCHANGED } \langle flag \rangle$$

$$EnterCriticalSection(p) \triangleq state[p] = \text{"}Waiting\text{"}$$
$$\land\ (flag[Not(p)] = \text{FALSE} \lor turn = p)$$
$$\land\ state' = [state \text{ EXCEPT } ![p] = \text{"}CriticalSection\text{"}]$$
$$\land\ \text{UNCHANGED } \langle turn, flag \rangle$$

$$ExitCriticalSection(p) \triangleq state[p] = \text{"}CriticalSection\text{""}$$
$$\land\ flag' = [flag \text{ EXCEPT } ![p] = \text{FALSE}]$$
$$\land\ state' = [state \text{ EXCEPT } ![p] = \text{"}Start\text{"}]$$
$$\land\ \text{UNCHANGED } \langle turn \rangle$$

MODULE *peterson_lock*

.....contd.

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

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

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

MODULE *peterson_lock*

.....contd.

$$
\begin{aligned}
&\mathit{ExecutionInvariant} \triangleq \forall i \in \mathit{ProcSet} : \text{''} \\
&\qquad \mathit{state}[i] \in \mathit{States} \setminus \{\text{''}\mathit{Start}\text{''}\} \implies \mathit{flag}[i] \\
&\qquad \land\ \mathit{state}[i] \in \{\text{''}\mathit{CriticalSection}\text{''}\} \implies \mathit{state}[\mathit{Not}(i)] \notin \{\text{''}\mathit{CriticalSection}\text{''}\} \\
&\qquad \land\ \mathit{state}[\mathit{Not}(i)] \in \{\text{''}\mathit{Waiting}\text{''}\} \implies \mathit{turn} = i
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{TypeInvariant} \triangleq \mathit{state} \in [\mathit{ProcSet} \rightarrow \mathit{States}] \\
&\qquad \land\ \mathit{turn} \in \mathit{ProcSet} \\
&\qquad \land\ \mathit{flag} \in [\mathit{ProcSet} \rightarrow \{\mathit{true}, \mathit{false}\}]
\end{aligned}
$$

$$\mathit{Inv} \triangleq \mathit{ExecutionInvariant} \land \mathit{TypeInvariant}$$

MODULE *peterson_lock*

.....contd.

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

$$\text{THEOREM } Spec \implies \Box MutualExclusion$$

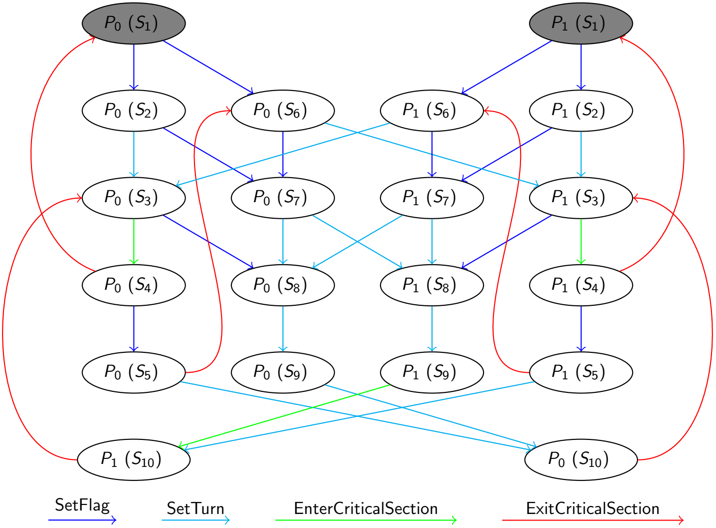
Model Checking Peterson's Algorithm

- Provide the model checker with predicates
- Model checker performs a state space exploration by successor calculation using the *Next* relation
- Checks if the predicate holds in the current state being explored
- Termination: only when a fix-point is reached.

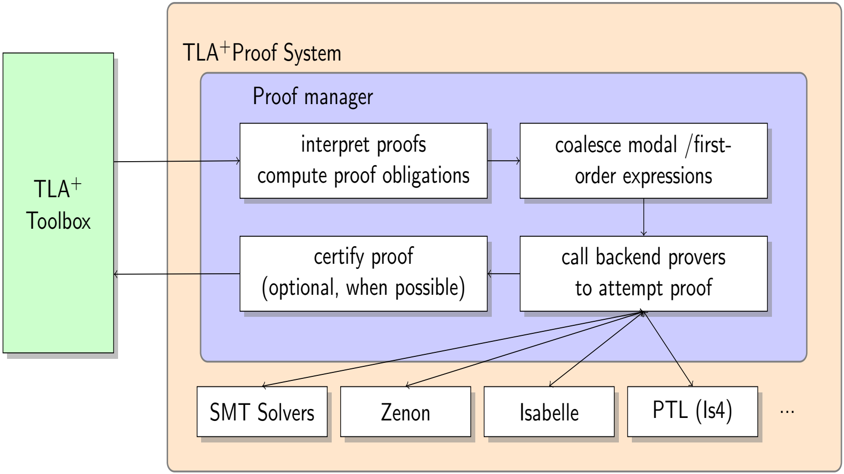
Is reaching a fixpoint decidable?

Turns out....No. Fortunately, we know our algorithm is correct so we can proceed.

Model Checking Peterson's Algorithm: Results



TLA$^+$Proof System Architecture



Proving Mutual Exclusion for Peterson's Algorithm in TLA$^+$

Proof Structure

- Structured as an **Invariance Proof**
- For every state in the system, prove that the invariant holds
- Use the invariant to prove that the desired property (here, mutual exclusion) holds for the system

Proving Mutual Exclusion for Peterson's Algorithm in TLA$^+$

Proof Outline

- Begin by proving that in the initial state, the invariant[2] holds
- Iterate through every possible state in the system and prove that the invariant holds in that state
- For every intermediate state of the system, prove that a state change does not violate the invariant
- Prove that the invariant implies mutual exclusion and thereby by a transitive relation, the system also implies mutual exclusion

[2] $Inv \triangleq ExecutionInvariant \land \; TypeInvariant$

Proving Mutual Exclusion for Peterson's Algorithm in TLA$^+$

Proof specification in TLA$^+$: Initial state

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

Proof Decomposition

$$\mathit{Init} \triangleq \mathit{flag} = [i \in \mathit{ProcSet} \mapsto \text{FALSE}\ ] \\ \land\ \mathit{turn} \in \{0, 1\} \\ \land\ \mathit{state} = [i \in \mathit{ProcSet} \mapsto \text{"}\mathit{Start}\text{"}]$$

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

$$\mathit{TypeInvariant} \triangleq \mathit{state} \in [\mathit{ProcSet} \to \mathit{States}] \\ \land\ \mathit{turn} \in \mathit{ProcSet} \\ \land\ \mathit{flag} \in [\mathit{ProcSet} \to \{\mathit{true}, \mathit{false}\}]$$

Proving Mutual Exclusion for Peterson's Algorithm in TLA$^+$

Proof specification in TLA$^+$: Initial state

$\text{THEOREM } \mathit{Spec} \implies \Box \mathit{MutualExclusion}$
PROOF
$\quad \langle 1 \rangle 1. \mathit{Init} \implies \mathit{Inv}$
$\quad\quad \text{BY DEFS } \mathit{Init}, \mathit{Inv}, \mathit{TypeInvariant}, \mathit{ExecutionInvariant}, \mathit{vars}, \mathit{States}, \mathit{ProcSet}$

Proof Decomposition

$$\begin{aligned}\mathit{Init} \triangleq\ & \mathit{flag} = [i \in \mathit{ProcSet} \mapsto \text{FALSE}\ ] \\ & \land\ \mathit{turn} \in \{0, 1\} \\ & \land\ \mathit{state} = [i \in \mathit{ProcSet} \mapsto \text{"}\mathit{Start}\text{"}]\end{aligned}$$

Using the definition of *Init* we can expand the Invariants. Since both processes have their state set to "*Start*" and their *flag* set to FALSE , we can replace the values in the *ExecutionInvariant* to get:

$$\begin{aligned}\mathit{ExecutionInvariant} \triangleq\ & \forall i \in \mathit{ProcSet} : \text{"} \\ & \text{"}\mathit{Start}\text{"} \in \mathit{States} \setminus \{\text{"}\mathit{Start}\text{"}\} \implies \text{FALSE} \\ & \land\ \text{"}\mathit{Start}\text{"} \in \{\text{"}\mathit{CriticalSection}\text{"}\} \implies \text{"}\mathit{Start}\text{"} \notin \{\text{"}\mathit{CriticalSection}\text{"}\} \\ & \land\ \text{"}\mathit{Start}\text{"} \in \{\text{"}\mathit{Waiting}\text{"}\} \implies \mathit{turn} = i\end{aligned}$$

$$\begin{aligned}\mathit{TypeInvariant} \triangleq\ & \mathit{state} \in [\mathit{ProcSet} \to \mathit{States}] \\ & \land\ \mathit{turn} \in \mathit{ProcSet} \\ & \land\ \mathit{flag} \in [\mathit{ProcSet} \to \{\mathit{true}, \mathit{false}\}]\end{aligned}$$

Proving Mutual Exclusion for Peterson's Algorithm in TLA$^+$

Proof specification in TLA$^+$: Intermediate States

$\langle 1 \rangle 2. \mathit{Inv} \land \; [\mathit{Next}]_{\mathit{vars}} \implies \mathit{Inv}'$
BY DEFS *Init*, *Inv*, *TypeInvariant*, *ExecutionInvariant*, *vars*, *States*, *ProcSet*

$\langle 2 \rangle 1.$SUFFICES ASSUME *Inv*, *Next* PROVE Inv'
BY DEFS *ExecutionInvariant*, *TypeInvariant*, *Inv*, *vars*

Proving Mutual Exclusion for Peterson's Algorithm in TLA$^+$

Proof specification in TLA$^+$: Intermediate States contd.

$\langle 2 \rangle 2.\, \mathit{TypeInvariant}'$
  BY $\langle 2 \rangle 1$
  DEFS *Inv*, *TypeInvariant*, *Next*, *proc*, *Not*, *States*, *ProcSet*, *SetFlag*, *SetTurn*, *EnterCriticalSection*, *ExitCriticalSection*

Proving Mutual Exclusion for Peterson's Algorithm in TLA$^+$

Proof specification in TLA$^+$: Intermediate States contd.

$\langle 2 \rangle 3.\mathit{ExecutionInvariant}'$

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

$\langle 3 \rangle 2.$PICK $i \in \mathit{ProcSet} : \mathit{proc}(i)$
BY $\langle 2 \rangle 1$ DEFS $\mathit{Next}, \mathit{ProcSet}, \mathit{proc}, \mathit{SetFlag}, \mathit{SetTurn}$,
$\mathit{EnterCriticalSection}, \mathit{ExitCriticalSection}$

$\langle 3 \rangle 3.$CASE $i = j$
BY $\langle 2 \rangle 1, \langle 3 \rangle 2$ DEFS $\mathit{ExecutionInvariant}, \mathit{TypeInvariant}, \mathit{Inv}, \mathit{proc}$,
$\mathit{Not}, \mathit{ProcSet}, \mathit{SetFlag}, \mathit{SetTurn}, \mathit{EnterCriticalSection}, \mathit{ExitCriticalSection}$

$\langle 3 \rangle 3.$CASE $i \neq j$
BY $\langle 2 \rangle 1, \langle 3 \rangle 2$ DEFS $\mathit{ExecutionInvariant}, \mathit{TypeInvariant}, \mathit{Inv}, \mathit{proc}, \mathit{Not}, \mathit{ProcSet}$,
$\mathit{SetFlag}, \mathit{SetTurn}, \mathit{EnterCriticalSection}, \mathit{ExitCriticalSection}$

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

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

$\langle 1 \rangle 3.$ *Inv* $\implies$ *MutualExclusion*

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

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

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

Special Mention

PTL is used by the back-end solvers to compute predicates containing Temporal operators.

Proving Mutual Exclusion for Peterson's Algorithm in TLA$^+$

Proof specification in TLA$^+$: Toolbox view

peterson_lock | Model_1

```
THEOREM Spec => []MutualExclusion
PROOF
    <1>1. Init => Inv
        BY DEFS Init, Inv, TypeInvariant, ExecutionInvariant, vars,States,ProcSet

    <1>2. Inv /\ [Next]_vars => Inv'
        <2>1. SUFFICES ASSUME Inv, Next PROVE Inv'
            BY DEFS ExecutionInvariant,TypeInvariant,Inv,vars
        <2>2. TypeInvariant'
            BY <2>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 <3>3 and <3>4
\***************************************************************************
        <2>3. ExecutionInvariant'
            <3>1. SUFFICES ASSUME NEW j \in ProcSet PROVE ExecutionInvariant!(j)' BY
            DEF ExecutionInvariant,ProcSet
            <3>2. PICK i \in ProcSet : proc(i)
                BY  <2>1
                DEF Next,ProcSet,proc, SetFlag, SetTurn,
                EnterCriticalSection, ExitCriticalSection
\***************************************************************************
\* If we select the same process and check the execution invariants, the proof is obvious
\***************************************************************************
            <3>3. CASE i=j
                BY <2>1,<3>2
                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.
\***************************************************************************
            <3>4. CASE i#j
                BY <2>1,<3>2
                DEFS ExecutionInvariant, TypeInvariant, Inv, proc, Not,ProcSet,
                 SetFlag, SetTurn, EnterCriticalSection, ExitCriticalSection
            <3>. QED BY <3>3,<3>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
\***************************************************************************
        <2>4. QED BY <2>2,<2>3 DEF Inv
```

Conclusion

- TLA$^+$can be used to specify fairly complex systems with ease.
- Verification of a specification becomes easier with the mathematical approach.
- Several stronger constructs can be used in practice to specify safety, liveness and fairness properties.
- Working with TLA$^+$requires some experience. It is not straightforward to understand and write a specification for a system.
- System specification and proofs can get fairly complex for considerably sized algorithms.

Thank you for your time.

Questions??

Dirk Beyer & Thomas Lemberger (2017): *Software Verification: Testing vs. Model Checking - A Comparative Evaluation of the State of the Art.*
In: *Haifa Verification Conference*, *Lecture Notes in Computer Science* 10629, Springer, pp. 99–114.

Prof. Joost-Pieter Katoen (2013): *Introduction to Model Checking*.
Available at `https://moves.rwth-aachen.de/wp-content/uploads/SS15/Introduction2MC/lec1.pdf`.

Appendix: PTL

Propositional Temporal Logic (PTL) introduces operators which have specific definitions for interpreting the variables at a given time any instant.

- $\circ A$: A holds at the time point immediately after the reference point. (*nexttime operator*).
- $\Box A$: A holds at all time points after the reference point. (*always or henceforth operator*).
- $\Diamond A$**:** There is a time point after the reference point at which A holds. *(sometime or eventually operator)*.
- A *atnext* B: A will hold at the next time point that B holds. (*first time or atnext operator*).
- A *until* B: A holds at all following time points up to a time point at which B holds. (*until operator*).

Remark

TLA$^+$proof system has a dedicated solver for temporal logic (LS4).