

This file contains the abstract program *UM* whose safety part is in Figure 4.2 of Section 4.2.2.1 of the book “A Science of Concurrent Programs” by *Leslie Lamport*.

The atomic action that describes execution of one step of process *p* beginning with control at a label such as *wait* is represented by the action *wait(p)*. The value of *pc(p)* indicating control is at *wait* is written as the string “wait” in the TLA+ version. (We could declare *wait* to be a constant and give another name to the action *wait(p)*, but it’s more convenient to use strings so we don’t have to add assumptions such as that *wait* doesn’t equal *ncs* .

As usual, *Init* and *Next* are the initial-state predicate and step predicate of the program.

EXTENDS *Integers*

VARIABLES *x, pc*

$v \triangleq \langle x, pc \rangle$

$Init \triangleq \wedge x = [p \in \{0, 1\} \mapsto \text{FALSE}]$   
 $\wedge pc = [p \in \{0, 1\} \mapsto \text{“ncs”}]$

$ncs(p) \triangleq \wedge pc[p] = \text{“ncs”}$   
 $\wedge \text{TRUE}$   
 $\wedge pc' = [pc \text{ EXCEPT } !p] = \text{“wait”}]$   
 $\wedge x' = x$

$wait(p) \triangleq \wedge pc[p] = \text{“wait”}$   
 $\wedge x' = [x \text{ EXCEPT } !p] = \text{TRUE}]$   
 $\wedge pc' = [pc \text{ EXCEPT } !p] = \text{“w2”}]$

$w2(p) \triangleq \wedge pc[p] = \text{“w2”}$   
 $\wedge \neg x[1 - p]$   
 $\wedge pc' = [pc \text{ EXCEPT } !p] = \text{“cs”}]$   
 $\wedge x' = x$

$cs(p) \triangleq \wedge pc[p] = \text{“cs”}$   
 $\wedge pc' = [pc \text{ EXCEPT } !p] = \text{“exit”}]$   
 $\wedge x' = x$

$exit(p) \triangleq \wedge pc[p] = \text{“exit”}$   
 $\wedge x' = [x \text{ EXCEPT } !p] = \text{FALSE}]$   
 $\wedge pc' = [pc \text{ EXCEPT } !p] = \text{“ncs”}]$

$PNext(p) \triangleq ncs(p) \vee wait(p) \vee w2(p) \vee cs(p) \vee exit(p)$

$Next \triangleq \exists p \in \{0, 1\} : PNext(p)$

$UMSafe \triangleq Init \wedge \Box [Next]_v$

Here are the definitions of *TypeOK* and the inductive invariant of *UM* given in the book, which we give the name *Inv*.

$$TypeOK \triangleq \wedge x \in [\{0, 1\} \rightarrow \text{BOOLEAN}] \\ \wedge pc \in [\{0, 1\} \rightarrow \{\text{"ncs"}, \text{"wait"}, \text{"w2"}, \text{"cs"}, \text{"exit"}\}]$$

$$Inv \triangleq \wedge TypeOK \\ \wedge \forall p \in \{0, 1\} : \wedge (pc[p] \in \{\text{"w2"}, \text{"cs"}\}) \Rightarrow x[p] \\ \wedge (pc[p] = \text{"cs"}) \Rightarrow (pc[1-p] \neq \text{"cs"})$$

We define *Mutex* to be formula (4.5) of the book, which is given there as the definition of mutual exclusion. More precisely, the algorithm *UM* satisfies mutual exclusion iff *Mutex* is an invariant of *UM*.

$$Mutex \triangleq \forall p, q \in \{0, 1\} : (p \neq q) \Rightarrow (\{pc[p], pc[q]\} \neq \{\text{"cs"}\})$$

*TLAPS*, the TLA+ proof checker, easily checks that *Inv* implies *Mutex*, so the invariance of *Inv* implies that *UM* satisfies mutual exclusion. The module *TLAPS* must be imported to use *TLAPS*.

INSTANCE *TLAPS*

THEOREM *Inv*  $\Rightarrow$  *Mutex*

BY DEF *TypeOK*, *Inv*, *Mutex*

The book defines *UMLive* to be formula (4.4), which is:

$$(\text{ENABLED } \langle PNext(p) \rangle_v) \leadsto \langle PNext(p) \rangle_v$$

However, the TLA+ grammar allows the TLA formula  $F \leadsto G$  only if  $F$  and  $G$  are both TLA+ temporal formula, which  $\langle PNext(p) \rangle_v$  isn't. So to write it, we must expand the definition of  $\leadsto$ .

$$UMLive \triangleq \forall p \in \{0, 1\} : \Box((\text{ENABLED } \langle PNext(p) \rangle_v) \Rightarrow \Diamond \langle PNext(p) \rangle_v)$$

The abstract program *UM* with its liveness property is described by the TLA+ formula  $UMSafe \wedge UMLive$ . This program is deadlock free, meaning that it satisfies this property:

$$DeadlockFree \triangleq (\exists p \in \{0, 1\} : pc[p] = \text{"wait"}) \\ \leadsto (\exists p \in \{0, 1\} : pc[p] = \text{"cs"})$$

However, *TLC* cannot check if  $UMSafe \wedge UMLive$  satisfies property *DeadlockFree* because it cannot check any liveness properties of a program with a liveness property of the form  $\Box(F \Rightarrow \Diamond \langle A \rangle_v)$ . (As explained in the book, we don't write programs with such a liveness property.)

\ \* Modification History  
 \ \* Last modified Mon Oct 14 10:47:39 CEST 2024 by lamport  
 \ \* Last modified Fri Oct 11 11:46:42 CEST 2024 by lblam  
 \ \* Created Sun Nov 05 10:32:54 CET 2023 by lamport