MODULE *Simple*

This is a trivial example from the document "Teaching *Conccurrency*" that appeared in

ACM SIGACT News Volume 40, Issue 1 (March 2009), 58 − 62

A copy of that article is at

http:=*lamport:azurewebsites:net*/pubs/teaching-*concurrency:pdf*

It is also the example in Section 7.2 of "Proving Safety Properties", which is at

http:=*lamport:azurewebsites:net*/tla/proving-*safety:pdf*

EXTENDS *Integers*, *TLAPS*

CONSTANT N

ASSUME $NAssump \triangleq (N \in Nat) \land (N > 0)$

****

Here is the algorithm in *PlusCal*. It's easy to understand if you think of the N processes, numbered from 0 through $N-1$, as arranged in a circle, with processes $(i-1)\%N$ and $(i+1)\%N$ being the processes on either side of process i.

```
--algorithm Simple f
    variables x = [i ∈ 0 .. (N − 1) ↦ 0], y = [i ∈ 0 .. (N − 1) ↦ 0] ;
    process ( proc ∈ 0 .. N − 1 ) f
      a: x[self] := 1 ;
      b: y[self] := x[(self − 1)%N]
     g
 g
```

****

BEGIN TRANSLATION This is the TLA+ translation of the *PlusCal* code.

VARIABLES x, y, pc

$vars \triangleq \langle x, y, pc \rangle$

$ProcSet \triangleq (0 \,..\, N-1)$

$Init \triangleq$ Global variables

$\land\ x = [i \in 0 \,..\, (N-1) \mapsto 0]$
$\land\ y = [i \in 0 \,..\, (N-1) \mapsto 0]$
$\land\ pc = [self \in ProcSet \mapsto \text{"a"}]$

$a(self) \triangleq \land\ pc[self] = \text{"a"}$
$\land\ x' = [x \text{ EXCEPT } ![self] = 1]$
$\land\ pc' = [pc \text{ EXCEPT } ![self] = \text{"b"}]$
$\land\ y' = y$

$b(self) \triangleq \land\ pc[self] = \text{"b"}$
$\land\ y' = [y \text{ EXCEPT } ![self] = x[(self-1)\%N]]$
$\land\ pc' = [pc \text{ EXCEPT } ![self] = \text{"Done"}]$
$\land\ x' = x$

$proc(self) \triangleq a(self) \lor b(self)$

$Next \triangleq (\exists\, self \in 0 \,..\, N-1 : proc(self))$
$\qquad \lor$ Disjunct to prevent deadlock on termination
$\qquad ((\forall\, self \in ProcSet : pc[self] = \text{“Done”}) \land \text{UNCHANGED } vars)$

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

$Termination \triangleq \Diamond(\forall\, self \in ProcSet : pc[self] = \text{“Done”})$

END TRANSLATION

The property of this algorithm we want to prove is that, when all the processses have terminated, $y[i]$ equals 1 for at least one process i. This property is express by the invariance of the following formula *PCorrect*. In other words, we have to prove the theorem

$Spec \Rightarrow \Box PCorrect$

$$PCorrect \triangleq (\forall\, i \in 0 \,..\, (N-1) : pc[i] = \text{“Done”}) \Rightarrow (\exists\, i \in 0 \,..\, (N-1) : y[i] = 1)$$

Proving the invariance of *PCorrect* requires finding an inductive invariant *Inv* that implies it. As usual, the inductive invariant includes a type-correctness invariant, which is the following formula *TypeOK*.

$$\begin{aligned} TypeOK \triangleq\ & \land x \in [0 \,..\, (N-1) \to \{0, 1\}] \\ & \land y \in [0 \,..\, (N-1) \to \{0, 1\}] \\ & \land pc \in [0 \,..\, (N-1) \to \{\text{“a”}, \text{“b”}, \text{“Done”}\}] \end{aligned}$$

It's easy to use *TLC* to check that the following formula *Inv* is an inductive invariant of the algorithm. You should also be able check that the propositional logic tautology

$(\mathrm{A} \Rightarrow B) = ((\neg A) \lor B)$

and the predicate logic tautology

$(\sim \forall\, i \in S : P(i)) = \exists\, i \in S : \sim P(i)$

imply that the last conjunct of *Inv* is equivalet to *PCorrect*. When I wrote the definition, I knew that this conjunct of *Inv* implied *PCorrect*, but I didn't realize that the two were equivalent until I saw the invariant written in terms of *PCorrect* in a paper by *Yuri Abraham*. That's why I originally didn't define *Inv* in terms of *PCorrect*. I'm not sure why, but I find the implication to be a more natural way to write the postcondition *PCorrect* and the disjunction to be a more natural way to think about the inductive invariant.

$$\begin{aligned} Inv \triangleq\ & \land TypeOK \\ & \land \forall\, i \in 0 \,..\, (N-1) : (pc[i] \in \{\text{“b”}, \text{“Done”}\}) \Rightarrow (x[i] = 1) \\ & \land \lor \exists\, i \in 0 \,..\, (N-1) : pc[i] \neq \text{“Done”} \\ & \quad \lor \exists\, i \in 0 \,..\, (N-1) : y[i] = 1 \end{aligned}$$

Here is the proof of correctness. The top-level steps $\langle 1 \rangle 1 - \langle 1 \rangle 4$ are the standard ones for an invariance proof, and the decomposition of the proof of $\langle 1 \rangle 2$ was done with the *Toolbox*'s Decompose Proof command. It was trivial to get *TLAPS* to check the proof, except for the proof of $\langle 2 \rangle 2$. A comment explains the problem I had with that step.

THEOREM $Spec \Rightarrow \Box PCorrect$

$\langle 1 \rangle$ USE *NAssump*

$\langle 1 \rangle 1.$ $Init \Rightarrow Inv$

BY DEF *Init*, *Inv*, *TypeOK*, *ProcSet*

$\langle 1 \rangle 2.$ $Inv \land [Next]_{vars} \Rightarrow Inv'$

$\langle 2 \rangle$ SUFFICES ASSUME Inv,
$[Next]_{vars}$
PROVE Inv'

OBVIOUS

$\langle 2 \rangle 1.$ ASSUME NEW $self \in 0 \,..\, (N-1)$,
$a(self)$
PROVE Inv'

BY $\langle 2 \rangle 1$ DEF *a*, *Inv*, *TypeOK*

$\langle 2 \rangle 2.$ ASSUME NEW $self \in 0 \,..\, (N-1)$,
$b(self)$
PROVE Inv'

I spent a lot of time decomposing this step down to about level $\langle 5 \rangle$ until I realized that the problem was that the default *SMT* solver in the version of *TLAPS* I was using was *CVC*3, which seems to know nothing about the % operator. When I used *Z*3, no further decomposition was needed.

BY $\langle 2 \rangle 2$, *Z*3 DEF *b*, *Inv*, *TypeOK*

$\langle 2 \rangle 3.$CASE UNCHANGED *vars*

BY $\langle 2 \rangle 3$ DEF *TypeOK*, *Inv*, *vars*

$\langle 2 \rangle 4.$ QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $\langle 2 \rangle 3$ DEF *Next*, *proc*

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

BY DEF *Inv*, *TypeOK*, *PCorrect*

$\langle 1 \rangle 4.$ QED

BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$, $\langle 1 \rangle 3$, *PTL* DEF *Spec*

* Modification History
* Last modified *Wed* May 15 02:33:18 *PDT* 2019 by *lamport*
* Created *Mon Apr* 15 16:25:14 *PDT* 2019 by *lamport*