MODULE *OneBitTwoProcesses*

EXTENDS *Integers*

```
***************************************************************
--algorithm OneBit
  { variable x = [i ∈ {0, 1} ↦ FALSE] ;
    fair process ( Proc ∈ {0, 1} )
    { ncs: while ( TRUE )
                      This is the model for my non-critical processing.
             {      skip ;
                 Ok, I'm done with that and I want in to the
                 critical section!
                e1: x[self] := TRUE ;
                e2: if ( ¬x[1 − self] )
                      If the other guy isn't in, I'm in!
                    { cs: skip  Model for my critical code! }
                    else
                      Oops, the other guy is in. What do I do?
                    { if ( self = 0 )
                        If I'm process 0, I'll keep trying.
                      { goto e2 }
                        But, if I'm process 1, I'll be the nice guy;
                        I'll stop trying and spin while process 0 is in.
                      else
                      { e3: x[1] := FALSE ;
                        e4: while ( x[0] )
                              { skip  spin } ;
                              goto e1
                     } } ;
                 Ok, I'm done. I don't need the critical section now.
                f:  x[self] := FALSE
             }
   } }
***************************************************************
```

BEGIN TRANSLATION

```
VARIABLES x, pc

vars ≜ ⟨x, pc⟩

ProcSet ≜ ({0, 1})

Init ≜  Global variables
        ∧ x = [i ∈ {0, 1} ↦ FALSE]
        ∧ pc = [self ∈ ProcSet ↦ "ncs"]

ncs(self) ≜  ∧ pc[self] = "ncs"
```

$$\begin{array}{l}
\land \text{TRUE} \\
\land pc' = [pc \text{ EXCEPT } ![self] = \text{“e1”}] \\
\land x' = x
\end{array}$$

$$e1(self) \triangleq \begin{array}[t]{l}
\land pc[self] = \text{“e1”} \\
\land x' = [x \text{ EXCEPT } ![self] = \text{TRUE}] \\
\land pc' = [pc \text{ EXCEPT } ![self] = \text{“e2”}]
\end{array}$$

$$e2(self) \triangleq \begin{array}[t]{l}
\land pc[self] = \text{“e2”} \\
\land \text{IF } \neg x[1 - self] \\
\qquad \text{THEN } \land pc' = [pc \text{ EXCEPT } ![self] = \text{“cs”}] \\
\qquad \text{ELSE } \land \text{IF } self = 0 \\
\qquad\qquad\qquad \text{THEN } \land pc' = [pc \text{ EXCEPT } ![self] = \text{“e2”}] \\
\qquad\qquad\qquad \text{ELSE } \land pc' = [pc \text{ EXCEPT } ![self] = \text{“e3”}] \\
\land x' = x
\end{array}$$

$$cs(self) \triangleq \begin{array}[t]{l}
\land pc[self] = \text{“cs”} \\
\land \text{TRUE} \\
\land pc' = [pc \text{ EXCEPT } ![self] = \text{“f”}] \\
\land x' = x
\end{array}$$

$$e3(self) \triangleq \begin{array}[t]{l}
\land pc[self] = \text{“e3”} \\
\land x' = [x \text{ EXCEPT } ![1] = \text{FALSE}] \\
\land pc' = [pc \text{ EXCEPT } ![self] = \text{“e4”}]
\end{array}$$

$$e4(self) \triangleq \begin{array}[t]{l}
\land pc[self] = \text{“e4”} \\
\land \text{IF } x[0] \\
\qquad \text{THEN } \land \text{TRUE} \\
\qquad\qquad\quad\; \land pc' = [pc \text{ EXCEPT } ![self] = \text{“e4”}] \\
\qquad \text{ELSE } \land pc' = [pc \text{ EXCEPT } ![self] = \text{“e1”}] \\
\land x' = x
\end{array}$$

$$f(self) \triangleq \begin{array}[t]{l}
\land pc[self] = \text{“f”} \\
\land x' = [x \text{ EXCEPT } ![self] = \text{FALSE}] \\
\land pc' = [pc \text{ EXCEPT } ![self] = \text{“ncs”}]
\end{array}$$

$$Proc(self) \triangleq \begin{array}[t]{l}
ncs(self) \lor e1(self) \lor e2(self) \lor cs(self) \lor e3(self) \\
\quad \lor e4(self) \lor f(self)
\end{array}$$

$$Next \triangleq (\exists\, self \in \{0, 1\} : Proc(self))$$

$$Spec \triangleq \begin{array}[t]{l}
\land Init \land \Box[Next]_{vars} \\
\land \forall\, self \in \{0, 1\} : \text{WF}_{vars}(Proc(self))
\end{array}$$

END TRANSLATION

Question 7.6

Analyzing weak fairness: Candidate Definition 1: Action A is weakly fair in behavior B if

$\neg\exists$ suffix of B : $\forall$ state $\in B$ A is enabled $\wedge \neg\exists\, A$ step.
$\equiv$
$\forall$ suffix of B : ($\exists$ state $\in B$: A is disabled) $\vee$ ($\exists\, A$ step).

we define $\langle A \rangle\_vars\ as\ a\ non - stuttering\ step : a\ step\ that\ changes\ at\ least\ one\ of\ the\ variables\ in$ “$vars$.”

WF$_vars(A) is\ satisfied\ by\ behavior\ B\ if\ every\ suffix\ of\ B\ has\ either\ a\ non - stuttering \langle A \rangle_vars\ step\ or\ at\ least\ one\ state\ where\ \langle A \rangle_vars\ is\ disabled.$

$\langle Proc(i) \rangle\_vars\ is\ a\ non - stuttering\ step\ and\ it's\ disabled\ for\ every\ suffix\ of\ a\ deadlocked\ behavior\ because\ there\ does\ not\ exist\ a\ non - stuttering\ action\ of\ Proc(i). Therefore, Proc(i) is\ weakly\ fair\ for\ the\ deadlocked\ behavior.$

$PC0Labels \triangleq \{$“ncs”, “f”, “e1”, “e2”, “cs”$\}$
$ExtraLabels \triangleq \{$“e3”, “e4”$\}$
$PC1Labels \triangleq PC0Labels \cup ExtraLabels$

$TypeOK \triangleq$ $\wedge\, pc[0] \in PC0Labels$
$\wedge\, pc[1] \in PC1Labels$
$pc \in [\{0, 1\} \rightarrow PC0Labels]$

I would like to have a more precise $TypeOK$, but don’t know how to write it. My more precise one would not let pc be “e3” or “e4.” The hyperbook’s solution is to add a conjunct to the inductive invariant that prevents this case (see “Inv” below).

$\wedge\, pc \in [\{0, 1\} \rightarrow \{$“ncs”, “f”, “e1”, “e2”, “e3”, “e4”, “cs”$\}]$
$\wedge\, x \in [\{0, 1\} \rightarrow$ BOOLEAN $]$

$InCS(i) \triangleq pc[i] =$ “cs”

$MutualExclusion \triangleq \neg(InCS(0) \wedge InCS(1))$

$Inv \triangleq$ $\wedge\, Init$
$\wedge\, TypeOK$
$\wedge\, MutualExclusion$
$\wedge\, pc[0] \notin \{$“e3”, “e4”$\}$
$\wedge\, \forall\, i \in \{0, 1\} : \mathrm{WF}_{vars}(Proc(i))$
$\wedge\, \forall\, i \in \{0, 1\} : InCS(i) \vee (pc[i] =$ “e2”$) \Rightarrow x[i]$

$ISpec \triangleq Inv \wedge \Box[Next]_{\langle x,\, pc \rangle}$

If we check $ISpec$ in this algorithm and only $Spec$ in that protocol, the checker will generate states that cannot be properly mapped to protocol states, such as this one:

Property line 134, col 12 to line 134, col 42 of module $OneBitProtocol$ is violated by the initial state:
$\wedge\, x =$ (0 :> FALSE @@ 1 :> FALSE)
$\wedge\, pc =$ (0 :> “ncs” @@ 1 :> “e1”)

If we check $Spec$ in this algorithm, we may check $A!Spec$ or $A!ISpec$ or both in the protocol. If we check $ISpec$ in this algorithm, we may not check $A!Spec$ in the protocol.

$$A \triangleq \textsc{instance } \mathit{OneBitProtocol}$$
$$\qquad \textsc{with } pc \leftarrow [i \in \{0, 1\} \mapsto$$
$$\qquad\quad \textsc{if } pc[i] \in \{\text{"ncs"}, \text{"f"}\} \textsc{ then } \text{"r"} \textsc{ else } pc[i]]$$

$$\mathit{Trying} \triangleq \land pc[0] \in \{\text{"e1"}, \text{"e2"}\}$$
$$\qquad\qquad \land pc[1] \in \{\text{"e1"}, \text{"e2"}\}$$

$$\mathit{Trying}(i) \triangleq pc[i] \in \{\text{"e1"}, \text{"e2"}\}$$

$$\mathit{DeadlockFree} \triangleq (\mathit{Trying}(0) \lor \mathit{Trying}(1)) \leadsto (\mathit{InCS}(0) \lor \mathit{InCS}(1))$$

* Modification History

* Last modified *Fri Feb* 21 19:48:24 *PST* 2014 by *bbeckman*

* Created *Thu Feb* 20 13:10:58 *PST* 2014 by *bbeckman*