MODULE $t2pc$

EXTENDS *Integers*, *Sequences*, *FiniteSets*, *TLC*
CONSTANT RM, The set of participating resource managers
$RMMAYFAIL$,
$TMMAYFAIL$,
$BTMENABLE$ TRUE when backup is needed, FALSE otherwise. Used to check problem 1.1

```
--algorithm TransactionCommit{
  variable rmState = [rm ∈ RM ↦ "working"],   initially all RMs are working
           tmState = "init" ;   TM is in "init" state initially
           btmState = "init" ;   BTM is also ready to take over if TM fails.

  define {
     The TM or BTM can only commit when rmState is prepared, committed or failed.
    canCommit ≜  ∀ rm ∈ RM : rmState[rm] ∈ {"prepared", "committed", "failed"}

     The TM or BTM can only abort when no other RM is committed.
    canAbort ≜  ∀ rm ∈ RM : rmState[rm] ≠ "committed"   TM can abort when no RM is in committed state


  }
  macro Prepare( p )
  {

       await rmState[p] ∈ {"working", "prepared"} ;
       rmState[p] := "prepared" ;

  }

  macro Decide( p )
  {

if ( tmState ≠ "hidden" )   If TM state is not hidden then execute this
      {
          either
              {
                  when  tmState = "commit" ;
                  if ( rmState[p] = "prepared" ) rmState[p] := "committed"

               }
          or
              {
                  when tmState = "abort" ;
                  if ( rmState[p] ∈ {"prepared", "working"} ) rmState[p] := "aborted"
               }
          or
              {
```

```
                    when rmState[p] = "working" ;
                    rmState[p] := "aborted"
                }
        }
else                                    If TM state is hidden then check for BTM state.
    {
        if ( BTMENABLE )
        {
          either
              {
                  when   btmState = "commit" ;

                  if ( rmState[p]   = "prepared" ) rmState[p] := "committed"

              }
          or
              {  When BTM state is "abort", then RM state goes to "aborted"
                  when btmState = "abort" ;
                  if ( rmState[p] ∈ {"prepared", "working"} ) rmState[p] := "aborted"
              }
          or
              {  RM can spontaneously abort when it's in the working state.
                  when rmState[p] = "working" ;
                  rmState[p] := "aborted"
              }
         }
       }


 }
 macro Fail( p )
  {
     if ( RMMAYFAIL )
     {
          rmState[p] := "failed"
      }
   }

fair process ( RManager ∈ RM )
{
    start: while ( rmState[self] ∈ {"working", "prepared"} ) {
    either Prepare(self)or Decide(self)or Fail(self) }
  }
fair process ( TManager = 0 )
```

```
{

    TS: either
        {
            await canCommit ;
            TC: if ( tmState ≠ "hidden" ∧ canCommit ) { tmState := "commit" } ;

            F1: if ( TMMAYFAIL )
                    {
                        Change the tmState to hidden only if it's not already hidden (does not really matter)
                        if ( tmState ≠ "hidden" ) {

                        Transfer the TM's state to BTM before going to hidden.
                        if ( BTMENABLE ) btmState := tmState ;
                        tmState := "hidden" ;

                        }

                    }
        }

        or
        {
            await canAbort ;
            TA: if ( tmState ≠ "hidden" ∧ canAbort ) { tmState := "abort" } ;

            F2: if ( TMMAYFAIL )
                    {
                        if ( tmState ≠ "hidden" )
                        {
                            Transfer BTM state to TM
                            if ( BTMENABLE ) btmState := tmState ;
                            tmState := "hidden" ;   TM fails

                        }

                    }
        }

  }

fair process ( BTManager = 10 )
```

```
{
  BTS: either
                {
                    await canCommit ;

                     BTM can commit only when BTM is enabled by the user, tmState is hidden and canCommit is true
                    BTC:  if ( tmState = "hidden" ∧ BTMENABLE ∧ canCommit )
                    {
                        print ⟨"committing when rmstate is", rmState⟩ ;
                        btmState := "commit" ;
                     }
                 }

                or
                {
                    await canAbort ;
                    BTA: if ( tmState = "hidden" ∧ BTMENABLE ∧ canAbort )
                     {

                      btmState := "abort" ;

                       }
                 }

  }
}
```

BEGIN TRANSLATION

VARIABLES $rmState$, $tmState$, $btmState$, pc

define statement

$canCommit \triangleq \forall\, rm \in RM : rmState[rm] \in \{\text{"prepared"}, \text{"committed"}, \text{"failed"}\}$

$canAbort \triangleq \forall\, rm \in RM : rmState[rm] \neq \text{"committed"}$

$vars \triangleq \langle rmState, tmState, btmState, pc\rangle$

$ProcSet \triangleq (RM) \cup \{0\} \cup \{10\}$

$Init \triangleq$ Global variables

$\wedge\, rmState = [rm \in RM \mapsto \text{"working"}]$
$\wedge\, tmState = \text{"init"}$
$\wedge\, btmState = \text{"init"}$
$\wedge\, pc = [self \in ProcSet \mapsto \text{CASE } self \in RM \rightarrow \text{"start"}$
$\qquad\qquad\qquad\qquad\qquad\qquad \square \quad self = 0 \rightarrow \text{"TS"}$
$\qquad\qquad\qquad\qquad\qquad\qquad \square \quad self = 10 \rightarrow \text{"BTS"}]$

$start(self) \triangleq \land pc[self] = \text{“start”}$
$\land \text{IF } rmState[self] \in \{\text{“working”}, \text{“prepared”}\}$
$\text{THEN } \land \lor \land rmState[self] \in \{\text{“working”}, \text{“prepared”}\}$
$\land rmState' = [rmState \text{ EXCEPT } ![self] = \text{“prepared”}]$
$\lor \land \text{IF } tmState \neq \text{“hidden”}$
$\text{THEN } \land \lor \land tmState = \text{“commit”}$
$\land \text{IF } rmState[self] = \text{“prepared”}$
$\text{THEN } \land rmState' = [rmState \text{ EXCEPT } ![self] = \text{“comm}$
$\text{ELSE } \land \text{TRUE}$
$\land \text{UNCHANGED } rmState$
$\lor \land tmState = \text{“abort”}$
$\land \text{IF } rmState[self] \in \{\text{“prepared”}, \text{“working”}\}$
$\text{THEN } \land rmState' = [rmState \text{ EXCEPT } ![self] = \text{“abort}$
$\text{ELSE } \land \text{TRUE}$
$\land \text{UNCHANGED } rmState$
$\lor \land rmState[self] = \text{“working”}$
$\land rmState' = [rmState \text{ EXCEPT } ![self] = \text{“aborted”}]$
$\text{ELSE } \land \text{IF } BTMENABLE$
$\text{THEN } \land \lor \land btmState = \text{“commit”}$
$\land \text{IF } rmState[self] = \text{“prepared”}$
$\text{THEN } \land rmState' = [rmState \text{ EXCEPT } ![s$
$\text{ELSE } \land \text{TRUE}$
$\land \text{UNCHANGED } rmState$
$\lor \land btmState = \text{“abort”}$
$\land \text{IF } rmState[self] \in \{\text{“prepared”}, \text{“working”}\}$
$\text{THEN } \land rmState' = [rmState \text{ EXCEPT } ![s$
$\text{ELSE } \land \text{TRUE}$
$\land \text{UNCHANGED } rmState$
$\lor \land rmState[self] = \text{“working”}$
$\land rmState' = [rmState \text{ EXCEPT } ![self] = \text{“abort}$
$\text{ELSE } \land \text{TRUE}$
$\land \text{UNCHANGED } rmState$
$\lor \land \text{IF } RMMAYFAIL$
$\text{THEN } \land rmState' = [rmState \text{ EXCEPT } ![self] = \text{“failed”}]$
$\text{ELSE } \land \text{TRUE}$
$\land \text{UNCHANGED } rmState$
$\land pc' = [pc \text{ EXCEPT } ![self] = \text{“start”}]$
$\text{ELSE } \land pc' = [pc \text{ EXCEPT } ![self] = \text{“Done”}]$
$\land \text{UNCHANGED } rmState$
$\land \text{UNCHANGED } \langle tmState, btmState \rangle$

$RManager(self) \triangleq start(self)$

$TS \triangleq \land pc[0] = \text{“TS”}$
$\land \lor \land canCommit$

```
                ∧ pc′ = [pc EXCEPT ![0] = "TC"]
              ∨ ∧ canAbort
                ∧ pc′ = [pc EXCEPT ![0] = "TA"]
            ∧ UNCHANGED ⟨rmState, tmState, btmState⟩

TC ≜  ∧ pc[0] = "TC"
      ∧ IF tmState ≠ "hidden" ∧  canCommit
            THEN ∧ tmState′ = "commit"
            ELSE ∧ TRUE
                 ∧ UNCHANGED tmState
      ∧ pc′ = [pc EXCEPT ![0] = "F1"]
      ∧ UNCHANGED ⟨rmState, btmState⟩

F1 ≜  ∧ pc[0] = "F1"
      ∧ IF TMMAYFAIL
            THEN ∧ IF tmState ≠ "hidden"
                       THEN ∧ IF BTMENABLE
                                  THEN ∧ btmState′ = tmState
                                  ELSE ∧ TRUE
                                       ∧ UNCHANGED btmState
                            ∧ tmState′ = "hidden"
                       ELSE ∧ TRUE
                            ∧ UNCHANGED ⟨tmState, btmState⟩
            ELSE ∧ TRUE
                 ∧ UNCHANGED ⟨tmState, btmState⟩
      ∧ pc′ = [pc EXCEPT ![0] = "Done"]
      ∧ UNCHANGED rmState

TA ≜  ∧ pc[0] = "TA"
      ∧ IF tmState ≠ "hidden" ∧ canAbort
            THEN ∧ tmState′ = "abort"
            ELSE ∧ TRUE
                 ∧ UNCHANGED tmState
      ∧ pc′ = [pc EXCEPT ![0] = "F2"]
      ∧ UNCHANGED ⟨rmState, btmState⟩

F2 ≜  ∧ pc[0] = "F2"
      ∧ IF TMMAYFAIL
            THEN ∧ IF tmState ≠ "hidden"
                       THEN ∧ IF BTMENABLE
                                  THEN ∧ btmState′ = tmState
                                  ELSE ∧ TRUE
                                       ∧ UNCHANGED btmState
                            ∧ tmState′ = "hidden"
                       ELSE ∧ TRUE
                            ∧ UNCHANGED ⟨tmState, btmState⟩
```

$$
\begin{array}{l}
\quad \text{ELSE} \quad \land \text{TRUE} \\
\qquad\qquad \land \text{UNCHANGED } \langle tmState,\ btmState \rangle \\
\land\ pc' = [pc \text{ EXCEPT } ![0] = \text{“Done”}] \\
\land \text{UNCHANGED } rmState
\end{array}
$$

$$TManager \triangleq TS \lor TC \lor F1 \lor TA \lor F2$$

$$
\begin{array}{l}
BTS \triangleq \land\ pc[10] = \text{“BTS”} \\
\qquad \land\ \lor\ \land\ canCommit \\
\qquad\qquad\quad \land\ pc' = [pc \text{ EXCEPT } ![10] = \text{“BTC”}] \\
\qquad\qquad \lor\ \land\ canAbort \\
\qquad\qquad\quad \land\ pc' = [pc \text{ EXCEPT } ![10] = \text{“BTA”}] \\
\qquad \land \text{UNCHANGED } \langle rmState,\ tmState,\ btmState \rangle
\end{array}
$$

$$
\begin{array}{l}
BTC \triangleq \land\ pc[10] = \text{“BTC”} \\
\qquad \land \text{IF } tmState = \text{“hidden”} \land BTMENABLE \land canCommit \\
\qquad\qquad \text{THEN } \land\ PrintT(\langle \text{“committing when rmstate is”},\ rmState \rangle) \\
\qquad\qquad\qquad\quad \land\ btmState' = \text{“commit”} \\
\qquad\qquad \text{ELSE } \land \text{TRUE} \\
\qquad\qquad\qquad\quad \land \text{UNCHANGED } btmState \\
\qquad \land\ pc' = [pc \text{ EXCEPT } ![10] = \text{“Done”}] \\
\qquad \land \text{UNCHANGED } \langle rmState,\ tmState \rangle
\end{array}
$$

$$
\begin{array}{l}
BTA \triangleq \land\ pc[10] = \text{“BTA”} \\
\qquad \land \text{IF } tmState = \text{“hidden”} \land BTMENABLE \land canAbort \\
\qquad\qquad \text{THEN } \land\ btmState' = \text{“abort”} \\
\qquad\qquad \text{ELSE } \land \text{TRUE} \\
\qquad\qquad\qquad\quad \land \text{UNCHANGED } btmState \\
\qquad \land\ pc' = [pc \text{ EXCEPT } ![10] = \text{“Done”}] \\
\qquad \land \text{UNCHANGED } \langle rmState,\ tmState \rangle
\end{array}
$$

$$BTManager \triangleq BTS \lor BTC \lor BTA$$

$$
\begin{array}{l}
Next \triangleq\ TManager \quad \lor BTManager \\
\qquad\quad \lor\ (\exists\, self \in RM : RManager(self)) \\
\qquad\quad \lor\ \text{Disjunct to prevent deadlock on termination} \\
\qquad\qquad (\forall\, self \in ProcSet : pc[self] = \text{“Done”}) \land \text{UNCHANGED } vars)
\end{array}
$$

$$
\begin{array}{l}
Spec \triangleq\ \land\ Init \land \Box[Next]_{vars} \\
\qquad\quad \land\ \forall\, self \in RM : \mathrm{WF}_{vars}(RManager(self)) \\
\qquad\quad \land\ \mathrm{WF}_{vars}(TManager) \\
\qquad\quad \land\ \mathrm{WF}_{vars}(BTManager)
\end{array}
$$

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

END TRANSLATION

Consistency property for *RMs*. Two *RMs* cannot be in aborted and committed state simultaneously.

$$ConsistentRM \triangleq$$

A state predicate asserting that two *RMs* have not arrived at conflicting decisions.

$$\forall\, rm1,\, rm2 \in RM : \neg \wedge rmState[rm1] = \text{“aborted”} \\ \wedge rmState[rm2] = \text{“committed”}$$

When the *TM* is active, then this needs to be satisfied for consistency

$$ConsistentTM \triangleq ((\neg tmState = \text{“hidden”}) \wedge (\forall\, rm \in RM : \neg \wedge rmState[rm] = \text{“committed”} \wedge tmState = \text{“ab} \\ \vee (tmState = \text{“hidden”} \wedge (\neg BTMENABLE))$$

When *TM* is not active, *i.e.* in hidden state, then this property needs to be satisfied.

$$ConsistentBTM \triangleq (BTMENABLE \wedge tmState = \text{“hidden”}) \wedge (\forall\, rm \in RM : \neg \wedge rmState[rm] = \text{“committed”}$$

This will be the overall consistent property taking into account the *RMs*, *TM* and *BTM*.

$$Consistent \triangleq ConsistentRM \wedge (ConsistentTM \vee ConsistentBTM)$$

Details of group members: Nitin *Nataraj* - nitinnat@*buffalo.edu*, 50246850 Arun *Krishnamurthy* - arunkris@*buffalo.edu*, 50247445

1.1 When *RMMAYFAIL* and *TMMAYFAIL* are both false, then the program runs with no errors. When *RMMAYFAIL* is true and *TMMAYFAIL* is false, the program still runs with no errors. This is correct because even though some *RMs* fail, the *TM* will look if other *RMs* have all committed or aborted.

1.2 When *RMMAYFAIL* is false and *TMMAYFAIL* is true, the temporal property is violated, *i.e.* Termination is not satisfied. On examining the stack trace, we see that when a state of $<$ *prepared*, aborted, *aborted* $>$ is reached and the *TM* fails (becomes hidden), the state does not change for a few more iterations. This is because there is no transaction manager to handle the *RM* requests. Since there is no way this state will change, it violates the termination property.

1.3 When both *TM* and *RM* are allowed to fail, and the *BTM* is enabled, then the program reaches termination and also does not violate consistency. When both *TM* and *RM* are both true and *BTM* is enabled, then the program still reaches termination since if *TM* fails, the *BTM* takes over and makes sure that the program reaches termination