```
MODULE AsyncFinishReplication
EXTENDS Integers

CONSTANTS CLIENT_NUM,          the number of clients
          MAX_KILL             maximum allowed kill events

VARIABLES state,               the program state, running or terminated
          clients,             clients sending value update requests to
                               master and backup
          master,              pool of master instances, only one is active
          backup,              pool of backup instances, only one is active
          msgs,                in-flight messages
          killed,              number of invoked kill actions to master or
                               backup
          tmp

Vars ≜ ⟨tmp, state, clients, master, backup, msgs, killed⟩
─────────────────────────────────────────────────────────────
C ≜ INSTANCE Commons
─────────────────────────────────────────────────────────────
TypeOK ≜
  Variables type constrains
  ∧ clients ∈ [C!CLIENT_ID → C!Client]
  ∧ master ∈ [C!INSTANCE_ID → C!Master]
  ∧ backup ∈ [C!INSTANCE_ID → C!Backup]
  ∧ state ∈ {"running", "terminated", "fatal"}
  ∧ msgs ⊆ C!Messages
  ∧ killed ∈ 0 .. MAX_KILL

StateOK ≜
  State invariants:
  - master version ≥ backup version
  - upon termination, the final version = the number of clients
  - if a fatal error occured, this must indicate the failure of both the master and the backup known
    by the client
  LET curMaster ≜ C!LastKnownMaster
      curBackup ≜ C!LastKnownBackup
  IN  ∧ curMaster.version ≥ curBackup.version
      ∧ IF state = "terminated"
          THEN ∧ curMaster.version = CLIENT_NUM
               ∧ curBackup.version = CLIENT_NUM
          ELSE ∧ curMaster.version ≤ CLIENT_NUM
               ∧ curBackup.version ≤ CLIENT_NUM
      ∧ IF state = "fatal"
          THEN ∃ c ∈ C!CLIENT_ID :
                 ∧ clients[c].phase = C!PH2_COMPLETED_FATAL
```

```
                    $\land\ master[clients[c].masterId].status = C!INST\_STATUS\_LOST$
                    $\land$ IF $clients[c].backupId \neq C!UNKNOWN\_ID$
                         THEN $backup[clients[c].backupId].status = C!INST\_STATUS\_LOST$
                         ELSE TRUE
           ELSE TRUE
```

$MustTerminate \triangleq$

The program must terminate by having all clients complete their update actions on both master and backup

$\Diamond(state \in \{\text{"terminated"}, \text{"fatal"}\})$

$Init \triangleq$

Initialiaze variables

```
$\land\ state = \text{"running"}$
$\land\ clients = [i \in C!CLIENT\_ID \mapsto [id \mapsto i, phase \mapsto C!PH1\_PENDING,$
              $value \mapsto i, masterId \mapsto C!FIRST\_ID, backupId \mapsto C!UNKNOWN\_ID]]$
$\land\ backup = [i \in C!INSTANCE\_ID \mapsto$
              IF $i = C!FIRST\_ID$
               THEN $[id \mapsto C!FIRST\_ID, masterId \mapsto C!FIRST\_ID, status \mapsto C!INST\_STATUS\_ACTIV$
                      $value \mapsto 0, version \mapsto 0]$
               ELSE $[id \mapsto i, masterId \mapsto C!UNKNOWN\_ID, status \mapsto C!INST\_STATUS\_NULL,$
                      $value \mapsto 0, version \mapsto 0]]$
$\land\ master = [i \in C!INSTANCE\_ID \mapsto$
              IF $i = C!FIRST\_ID$
               THEN $[id \mapsto C!FIRST\_ID, backupId \mapsto C!FIRST\_ID, status \mapsto C!INST\_STATUS\_ACTIV$
                      $value \mapsto 0, version \mapsto 0]$
               ELSE $[id \mapsto i, backupId \mapsto C!UNKNOWN\_ID, status \mapsto C!INST\_STATUS\_NULL,$
                      $value \mapsto 0, version \mapsto 0]]$
$\land\ msgs = \{\}$
$\land\ killed = 0$
$\land\ tmp = \{\}$
```

$AtLeastOneClientStarted \triangleq$

We use this condition to prevent killing a master or backup before at least one client starts

```
$\lor\ \land\ killed > 0$
$\lor\ \land\ killed = 0$
   $\land\ \exists\, c \in C!CLIENT\_ID : clients[c].phase \neq C!PH1\_PENDING$
```

$KillMaster \triangleq$

Kill the active master instance.

```
$\land\ state = \text{"running"}$
$\land\ AtLeastOneClientStarted$
$\land\ killed < MAX\_KILL$
```

```
$\land$ LET $activeM \triangleq C!FindMaster(C!INST\_STATUS\_ACTIVE)$
  IN  $\land$ $activeM \neq C!NOT\_MASTER$
      $\land$ $master' = [master$ EXCEPT $![activeM.id].status = C!INST\_STATUS\_LOST]$
      $\land$ $killed' = killed + 1$
$\land$ UNCHANGED $\langle tmp, state, clients, backup, msgs \rangle$
```

$KillBackup \triangleq$

Kill the active backup instance.

```
$\land$ $state =$ "running"
$\land$ $AtLeastOneClientStarted$
$\land$ $killed < MAX\_KILL$
$\land$ LET $activeB \triangleq C!FindBackup(C!INST\_STATUS\_ACTIVE)$
  IN  $\land$ $activeB \neq C!NOT\_BACKUP$
      $\land$ $backup' = [backup$ EXCEPT $![activeB.id].status = C!INST\_STATUS\_LOST]$
      $\land$ $killed' = killed + 1$
$\land$ UNCHANGED $\langle tmp, state, clients, master, msgs \rangle$
```

$C\_Start \triangleq$

Client start the replication process by sending "do" to master

```
$\land$ $state =$ "running"
$\land$ LET $client \triangleq C!FindClient(C!PH1\_PENDING)$
  IN  $\land$ $client \neq C!NOT\_CLIENT$
      $\land$ $C!SendMsg([from \mapsto$ "c",
                      $to \mapsto$ "m",
                      $clientId \mapsto client.id,$
                      $masterId \mapsto client.masterId,$
                      $backupId \mapsto C!UNKNOWN\_ID,$
                      $value \mapsto client.value,$
                      $tag \mapsto$ "masterDo"$])$
      $\land$ $clients' = [clients$ EXCEPT $![client.id].phase = C!PH2\_WORKING]$
$\land$ UNCHANGED $\langle tmp, state, master, backup, killed \rangle$
```

$M\_HandleDo \triangleq$

Master receiving "do", updating value, and sending "done"

```
$\land$ $state =$ "running"
$\land$ LET $msg \triangleq C!FindMessageToWithTag($"m"$, C!INST\_STATUS\_ACTIVE,$ "masterDo"$)$
  IN  $\land$ $msg \neq C!NOT\_MESSAGE$
      $\land$ $master' = [master$ EXCEPT $![msg.masterId].value = master[msg.masterId].value + msg.value,$
                                $![msg.masterId].version = master[msg.masterId].version + 1]$
      $\land$ $C!ReplaceMsg(msg, [from \mapsto$ "m",
                             $to \mapsto$ "c",
                             $clientId \mapsto msg.clientId,$
                             $masterId \mapsto msg.masterId,$
                             $backupId \mapsto master[msg.masterId].backupId,$
                             $value \mapsto 0,$
```

```
                                        $tag \mapsto$ "masterDone"])
  $\land$ UNCHANGED $\langle tmp, \quad state, clients, backup, killed \rangle$

$C\_HandleMasterDone \triangleq$
 Client receiving "done" from master, and forwarding action to backup
  $\land$ $state =$ "running"
  $\land$ LET $msg \triangleq C!FindMessageToClient($"m", "masterDone"$)$
     IN    $\land$ $msg \neq C!NOT\_MESSAGE$
           $\land$ $C!ReplaceMsg(msg, [from \mapsto$ "c",
                                        $to \mapsto$ "b",
                                        $clientId \mapsto msg.clientId,$
                                        $masterId \mapsto msg.masterId,$
                                        $backupId \mapsto msg.backupId,$
                                        $value \mapsto clients[msg.clientId].value,$
                                        $tag \mapsto$ "backupDo"$])$
           update our knowledge about the backup identity
           $\land$ $clients' = [clients$ EXCEPT $![msg.clientId].backupId = msg.backupId]$
  $\land$ UNCHANGED $\langle tmp, \quad state, master, backup, killed \rangle$

$B\_HandleDo \triangleq$
 Backup receiving "do", updating value, then sending "done"
  $\land$ $state =$ "running"
  $\land$ LET $msg \triangleq C!FindMessageToWithTag($"b", $C!INST\_STATUS\_ACTIVE,$ "backupDo"$)$
     IN    $\land$ $msg \neq C!NOT\_MESSAGE$
           $\land$ IF $msg.masterId = backup[msg.backupId].masterId$
              THEN   Master info is consistent between client and backup
                     $\land$ $backup' = [backup$ EXCEPT $![msg.backupId].value = backup[msg.backupId].value + msg.v$
                                                      $![msg.backupId].version = backup[msg.backupId].version + 1]$
                     $\land$ $C!ReplaceMsg(msg, [from \mapsto$ "b",
                                                  $to \mapsto$ "c",
                                                  $clientId \mapsto msg.clientId,$
                                                  $masterId \mapsto msg.masterId,$
                                                  $backupId \mapsto msg.backupId,$
                                                  $value \mapsto 0,$
                                                  $tag \mapsto$ "backupDone"$])$
              ELSE   Master has changed, client must restart
                     $\land$ $backup' = backup$
                     $\land$ $C!ReplaceMsg(msg, [from \mapsto$ "b",
                                                  $to \mapsto$ "c",
                                                  $clientId \mapsto msg.clientId,$
                                                  $masterId \mapsto backup[msg.backupId].masterId,$
                                                  $backupId \mapsto msg.backupId,$
                                                  $value \mapsto 0,$
                                                  $tag \mapsto$ "newMasterId"$])$
  $\land$ UNCHANGED $\langle tmp, \quad state, clients, master, killed \rangle$
```

```
$C\_HandleBackupDone \triangleq$
  Client receiving "done" from backup. Replication completed
  $\land$ $state = $ "running"
  $\land$ LET $msg \triangleq C!FindMessageToClient($"b", "backupDone"$)$
     IN  $\land$ $msg \neq C!NOT\_MESSAGE$
         $\land$ $C!RecvMsg(msg)$
         $\land$ $clients' = [clients$ EXCEPT $![msg.clientId].phase = C!PH2\_COMPLETED]$
  $\land$ UNCHANGED $\langle tmp, \quad state, master, backup, killed \rangle$
```

```
$Sys\_NotifyMasterFailure \triangleq$
  System notifying client of a dead master
  $\land$ $state = $ "running"
  $\land$ LET $msg \triangleq C!FindMessageTo($"m"$, C!INST\_STATUS\_LOST)$
     IN  $\land$ $msg \neq C!NOT\_MESSAGE$
         $\land$ LET $notifyTag \triangleq$ IF $msg.tag = $ "masterDo"
                                  THEN "masterDoFailed"
                                  ELSE IF $msg.tag = $ "masterGetNewBackup"
                                  THEN "masterGetNewBackupFailed"
                                  ELSE "INVALID"  this should be unreachable
            IN  $\land$ $notifyTag \neq $ "INVALID"
                $\land$ $C!ReplaceMsg(msg,$
                      $[from \mapsto $ "sys",
                       $to \mapsto $ "c",
                       $clientId \ \ \mapsto msg.clientId,$
                       $masterId \mapsto C!UNKNOWN\_ID,$
                       $backupId \mapsto C!UNKNOWN\_ID,$
                       $value \mapsto 0,$
                       $tag \mapsto notifyTag])$
  $\land$ UNCHANGED $\langle tmp, \quad state, clients, master, backup, killed \rangle$

$Sys\_NotifyBackupFailure \triangleq$
  System notifying client of a dead backup
  $\land$ $state = $ "running"
  $\land$ LET $msg \triangleq C!FindMessageTo($"b"$, C!INST\_STATUS\_LOST)$
     IN  $\land$ $msg \neq C!NOT\_MESSAGE$
         $\land$ LET $notifyTag \triangleq$ IF $msg.tag = $ "backupDo"
                                  THEN "backupDoFailed"
                                  ELSE IF $msg.tag = $ "backupGetNewMaster"
                                  THEN "backupGetNewMasterFailed"
                                  ELSE "INVALID"  this should be unreachable
            IN  $\land$ $notifyTag \neq $ "INVALID"
                $\land$ $C!ReplaceMsg(msg,$
                      $[from \mapsto $ "sys",
                       $to \mapsto $ "c",
                  $clientId \mapsto msg.clientId,$
                  $masterId \mapsto C!UNKNOWN\_ID,$
                  $backupId \mapsto C!UNKNOWN\_ID,$
                  $value \mapsto 0,$
                  $tag \mapsto notifyTag])$
  $\land$ UNCHANGED $\langle tmp, state, clients, master, backup, killed \rangle$
```

```
$C\_HandleMasterDoFailed \triangleq$
  Client received the system's notification of a dead master, and is requesting the backup to return
  the new master info
  $\land state = $ "running"
  $\land$ LET $msg \triangleq C!FindMessageToClient($"sys", "masterDoFailed"$)$
          $knownBackup \triangleq$ IF $msg \neq C!NOT\_MESSAGE$
                                 THEN $C!SearchForBackup$
                                 ELSE $C!NOT\_BACKUP$
     IN  $\land msg \neq C!NOT\_MESSAGE$
         $\land$ IF $knownBackup = C!NOT\_BACKUP$
            THEN $\land C!RecvMsg(msg)$
                 $\land state' = $ "fatal"
                 $\land clients' = [clients$ EXCEPT $![msg.clientId].phase = C!PH2\_COMPLETED\_FATAL]$
            ELSE $\land C!ReplaceMsg(msg, [from \mapsto $ "c",
                                        $to \mapsto$ "b",
                                        $clientId \mapsto msg.clientId,$
                                         send the client's master knowledge,
                                         to force the backup to not respond until rereplication
                                        $masterId \mapsto clients[msg.clientId].masterId,$
                                        $backupId \mapsto knownBackup.id,$
                                        $value \mapsto 0,$
                                        $tag \mapsto$ "backupGetNewMaster"$])$
                 $\land state' = state$
                 $\land clients' = clients$
  $\land$ UNCHANGED $\langle tmp, master, backup, killed \rangle$

$C\_HandleBackupDoFailed \triangleq$
  Client received the system's notification of a dead backup, and is requesting the master to return
  the new backup info
  $\land state = $ "running"
  $\land$ LET $msg \triangleq C!FindMessageToClient($"sys", "backupDoFailed"$)$
     IN  $\land msg \neq C!NOT\_MESSAGE$
         $\land C!ReplaceMsg(msg, [from \mapsto $ "c",
                                 $to \mapsto$ "m",
                                 $clientId \mapsto msg.clientId,$
                                 $masterId \mapsto clients[msg.clientId].masterId,$
                                  send the client's backup knowledge,
                                  to force the master to not respond until rereplication
```

```
                                    $backupId \mapsto clients[msg.clientId].backupId$,
                                    $value \mapsto 0$,
                                    $tag \mapsto$ "masterGetNewBackup"])
  $\land$ UNCHANGED $\langle tmp, \quad state, clients, master, backup, killed \rangle$
```

```
$M\_HandleGetNewBackup \triangleq$
```

Master responding to client with updated backup identity

```
  $\land state =$ "running"
  $\land$ LET $msg \triangleq C!FindMessageToWithTag($"m", $C!INST\_STATUS\_ACTIVE$, "masterGetNewBackup"$)$
     IN   $\land msg \neq C!NOT\_MESSAGE$
              master must not respond until it recovers the dead backup
          $\land msg.backupId \neq master[msg.masterId].backupId$
          $\land C!ReplaceMsg(msg, [from \mapsto$ "m",
                                    $to \mapsto$ "c",
                                    $clientId \ \ \mapsto msg.clientId$,
                                    $masterId \mapsto msg.masterId$,
                                    $backupId \mapsto master[msg.masterId].backupId$,
                                    $value \mapsto 0$,
                                    $tag \mapsto$ "newBackupId"])
  $\land$ UNCHANGED $\langle tmp, \quad state, clients, master, backup, killed \rangle$
```

```
$B\_HandleGetNewMaster \triangleq$
```

Backup responding to client with updated master identity

```
  $\land state =$ "running"
  $\land$ LET $msg \triangleq C!FindMessageToWithTag($"b", $C!INST\_STATUS\_ACTIVE$, "backupGetNewMaster"$)$
     IN   $\land msg \neq C!NOT\_MESSAGE$
              backup must not respond until it recovers the dead master
          $\land msg.masterId \neq backup[msg.backupId].masterId$
          $\land C!ReplaceMsg(msg, [from \mapsto$ "b",
                                    $to \mapsto$ "c",
                                    $clientId \ \ \mapsto msg.clientId$,
                                    $masterId \mapsto backup[msg.backupId].masterId$,
                                    $backupId \mapsto msg.backupId$,
                                    $value \mapsto 0$,
                                    $tag \mapsto$ "newMasterId"])
  $\land$ UNCHANGED $\langle tmp, \quad state, clients, master, backup, killed \rangle$
```

```
$C\_HandleBackupGetNewMasterFailed \triangleq$
```

The client handling the failure of the backup, when the client asked the backup to return the new master identity. The client mannually searches for the master. If manual search does not find a master, a fatal error occurs. Otherwise, the client updates it's *masterId* and eventually restarts. Restarting is safe because this action is reached only if "masterDo" fails

```
  $\land state =$ "running"
  $\land$ LET $msg \triangleq C!FindMessageToClient($"sys", "backupGetNewMasterFailed"$)$
```

```
        $searchManually \triangleq msg \neq C!NOT\_MESSAGE$
        $foundMaster \triangleq C!SearchForMaster$
    IN  $\land msg \neq C!NOT\_MESSAGE$
        $\land searchManually$
        $\land C!RecvMsg(msg)$
        $\land$ IF $foundMaster = C!NOT\_MASTER$   no live master found
           THEN $\land state' = $ "fatal"
                $\land clients' = [clients$ EXCEPT $![msg.clientId].phase = C!PH2\_COMPLETED\_FATAL]$
           ELSE $\land state' = state$
                  at this point, the live master must have been changed
                $\land foundMaster.id \neq clients[msg.clientId].masterId$
                  change status to pending to be eligible for restart
                $\land clients' = [clients$ EXCEPT $![msg.clientId].masterId = foundMaster.id,$
                                                 $![msg.clientId].phase = C!PH1\_PENDING]$
  $\land$ UNCHANGED $\langle tmp, \quad master, backup, killed\rangle$

$C\_HandleMasterGetNewBackupFailed \triangleq$
  The client handling the failure of the master when the client asked the master to return the
  new backup identity. The failure of the master is fatal. If a recovered master exists we should
  not search for it, because it may have the old version before masterDone.
  $\land state = $ "running"
  $\land$ LET $msg \triangleq C!FindMessageToClient($"sys", "masterGetNewBackupFailed"$)$
    IN  $\land msg \neq C!NOT\_MESSAGE$
        $\land state' = $ "fatal"
        $\land clients' = [clients$ EXCEPT $![msg.clientId].phase = C!PH2\_COMPLETED\_FATAL]$
        $\land C!RecvMsg(msg)$
  $\land$ UNCHANGED $\langle tmp, \quad master, backup, killed\rangle$
```

```
$C\_UpdateBackupId \triangleq$
  $\land state = $ "running"
  $\land$ LET $msg \triangleq C!FindMessageToClient($"m", "newBackupId"$)$
    IN  $\land msg \neq C!NOT\_MESSAGE$   receive new backup identity, and complete request,
                                        don't restart, master is alive and up to date
        $\land C!RecvMsg(msg)$
        $\land clients' = [clients$ EXCEPT $![msg.clientId].backupId = msg.backupId,$
                                         $![msg.clientId].phase = C!PH2\_COMPLETED]$
  $\land$ UNCHANGED $\langle tmp, \quad state, master, backup, killed\rangle$

$C\_UpdateMasterIdAndRestart \triangleq$
  Client receiving a new master identify from a live backup and is preparing to restart
  $\land state = $ "running"
  $\land$ LET $msg \triangleq C!FindMessageToClient($"b", "newMasterId"$)$
    IN  $\land msg \neq C!NOT\_MESSAGE$
        $\land C!RecvMsg(msg)$
        ∧ clients′ = [clients EXCEPT ![msg.clientId].masterId = msg.masterId,
                                    ![msg.clientId].phase = C!PH1_PENDING]
    ∧ UNCHANGED ⟨tmp,   state, master, backup, killed⟩
────────────────────────────────────────────────────────────────────
M_DetectBackupLost ≜
  Master detected backup failure and is getting ready to recovery it
    ∧ state = "running"
    ∧ LET activeM ≜ C!FindMaster(C!INST_STATUS_ACTIVE)
          liveB ≜ C!LiveBackup
      IN  ∧ activeM ≠ C!NOT_MASTER   master is active
          ∧ liveB = C!NOT_BACKUP   backup is lost
          ∧ master′ = [master EXCEPT ![activeM.id].status = C!INST_STATUS_BUSY]
    ∧ UNCHANGED ⟨tmp,   state, clients, backup, msgs, killed⟩

M_RecoverBackup ≜
  Master creating a new backup using its own state. Master does not process any client requests
  during recovery
    ∧ state = "running"
    ∧ LET busyM ≜ C!FindMaster(C!INST_STATUS_BUSY)
          lostB  ≜ C!LastLostBackup
      IN  ∧ lostB  ≠ C!NOT_BACKUP   a lost backup exists
          ∧ busyM ≠ C!NOT_MASTER   master is busy recovering master
          ∧ LET newBackupId ≜ lostB.id + 1
            IN  ∧ newBackupId ≤ C!MAX_INSTANCE_ID
                ∧ backup′ = [backup EXCEPT ![newBackupId].status = C!INST_STATUS_ACTIVE,
                                           ![newBackupId].masterId = busyM.id,
                                           ![newBackupId].value = busyM.value,
                                           ![newBackupId].version = busyM.version]
                ∧ master′ = [master EXCEPT ![busyM.id].status = C!INST_STATUS_ACTIVE,
                                           ![busyM.id].backupId = newBackupId ]
    ∧ UNCHANGED ⟨tmp,   state, clients, msgs, killed⟩

B_DetectMasterLost ≜
  Backup detected master failure and is getting ready to recover it
    ∧ state = "running"
    ∧ LET liveM ≜ C!SearchForMaster
          activeB ≜ C!FindBackup(C!INST_STATUS_ACTIVE)
      IN  ∧ liveM = C!NOT_MASTER   master is not active
          ∧ activeB ≠ C!NOT_BACKUP   backup is active
          ∧ backup′ = [backup EXCEPT ![activeB.id].status = C!INST_STATUS_BUSY]
    ∧ UNCHANGED ⟨tmp,   state, clients, master, msgs, killed⟩

B_RecoverMaster ≜
  Backup creating a new master using its own state. Backup does not process any client requests
  during recovery
```

```
$\land$ $state = $ "running"
$\land$ LET $lostM \triangleq C!LastLostMaster$
        $busyB \triangleq C!FindBackup(C!INST\_STATUS\_BUSY)$
   IN  $\land$ $lostM \neq C!NOT\_MASTER$  a lost master exists
       $\land$ $busyB \neq C!NOT\_BACKUP$  backup is busy recovering master
       $\land$ LET $newMasterId \triangleq lostM.id + 1$
          IN  $\land$ $newMasterId \leq C!MAX\_INSTANCE\_ID$
              $\land$ $master' = [master$ EXCEPT $![newMasterId].status = C!INST\_STATUS\_ACTIVE,$
                                        $![newMasterId].backupId = busyB.id,$
                                        $![newMasterId].value = busyB.value,$
                                        $![newMasterId].version = busyB.version]$
              $\land$ $backup' = [backup$ EXCEPT $![busyB.id].status = C!INST\_STATUS\_ACTIVE,$
                                        $![busyB.id].masterId = newMasterId\ ]$
$\land$ UNCHANGED $\langle tmp,\ \ state, clients, msgs, killed \rangle$
```

```
$TerminateSuccessfully \triangleq$
  $TerminateSuccessfully$ the program if all clients completed their work
  $\land$ $state = $ "running"
      wait for all clients to complete updating the master and backup
  $\land$ $\forall c \in C!CLIENT\_ID : clients[c].phase = C!PH2\_COMPLETED$
  $\land$ $state' = $ "terminated"
  $\land$ UNCHANGED $\langle tmp,\ \ clients, master, backup, msgs, killed \rangle$

$Next \triangleq$
  $\lor$ $KillMaster$
  $\lor$ $KillBackup$
  $\lor$ $C\_Start$
  $\lor$ $M\_HandleDo$
  $\lor$ $C\_HandleMasterDone$
  $\lor$ $B\_HandleDo$
  $\lor$ $C\_HandleBackupDone$
  $\lor$ $Sys\_NotifyMasterFailure$
  $\lor$ $Sys\_NotifyBackupFailure$
  $\lor$ $C\_HandleMasterDoFailed$
  $\lor$ $C\_HandleBackupDoFailed$
  $\lor$ $M\_HandleGetNewBackup$
  $\lor$ $B\_HandleGetNewMaster$
  $\lor$ $C\_HandleBackupGetNewMasterFailed$
  $\lor$ $C\_HandleMasterGetNewBackupFailed$
  $\lor$ $C\_UpdateBackupId$
  $\lor$ $C\_UpdateMasterIdAndRestart$
  $\lor$ $M\_DetectBackupLost$
  $\lor$ $M\_RecoverBackup$
  $\lor$ $B\_DetectMasterLost$
  $\lor$ $B\_RecoverMaster$
```

$\lor$ *TerminateSuccessfully*

$$
\begin{aligned}
Liveness \triangleq\ & \\
&\land \mathrm{WF}_{Vars}(KillMaster)\\
&\land \mathrm{WF}_{Vars}(KillBackup)\\
&\land \mathrm{WF}_{Vars}(C\_Start)\\
&\land \mathrm{WF}_{Vars}(M\_HandleDo)\\
&\land \mathrm{WF}_{Vars}(C\_HandleMasterDone)\\
&\land \mathrm{WF}_{Vars}(B\_HandleDo)\\
&\land \mathrm{WF}_{Vars}(C\_HandleBackupDone)\\
&\land \mathrm{WF}_{Vars}(Sys\_NotifyMasterFailure)\\
&\land \mathrm{WF}_{Vars}(Sys\_NotifyBackupFailure)\\
&\land \mathrm{WF}_{Vars}(C\_HandleMasterDoFailed)\\
&\land \mathrm{WF}_{Vars}(C\_HandleBackupDoFailed)\\
&\land \mathrm{WF}_{Vars}(M\_HandleGetNewBackup)\\
&\land \mathrm{WF}_{Vars}(B\_HandleGetNewMaster)\\
&\land \mathrm{WF}_{Vars}(C\_HandleBackupGetNewMasterFailed)\\
&\land \mathrm{WF}_{Vars}(C\_HandleMasterGetNewBackupFailed)\\
&\land \mathrm{WF}_{Vars}(C\_UpdateBackupId)\\
&\land \mathrm{WF}_{Vars}(C\_UpdateMasterIdAndRestart)\\
&\land \mathrm{WF}_{Vars}(M\_DetectBackupLost)\\
&\land \mathrm{WF}_{Vars}(M\_RecoverBackup)\\
&\land \mathrm{WF}_{Vars}(B\_DetectMasterLost)\\
&\land \mathrm{WF}_{Vars}(B\_RecoverMaster)\\
&\land \mathrm{WF}_{Vars}(TerminateSuccessfully)
\end{aligned}
$$

Specification

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

THEOREM $Spec \Rightarrow \Box(TypeOK \land StateOK)$

\* Modification History
\* Last modified *Mon Mar* 19 20:30:28 *AEDT* 2018 by *u5482878*
\* Last modified Sat *Mar* 17 16:42:36 *AEDT* 2018 by *shamouda*
\* Created *Mon Mar* 05 13:44:57 *AEDT* 2018 by *u5482878*