```
┌──────────────── MODULE AsyncFinishReplication ────────────────┐
EXTENDS Integers

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

VARIABLES exec_state,            the execution state of the program: running, success, or fatal
          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

Vars ≜ ⟨exec_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]
  ∧ exec_state ∈ {"running", "success", "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 exec_state = "success"
          THEN ∧ curMaster.version = CLIENT_NUM
               ∧ curBackup.version = CLIENT_NUM
          ELSE ∧ curMaster.version ≤ CLIENT_NUM
               ∧ curBackup.version ≤ CLIENT_NUM
      ∧ IF exec_state = "fatal"
          THEN ∃ c ∈ C!CLIENT_ID :
                  ∧ clients[c].phase = C!PH2_COMPLETED_FATAL
                  ∧ master[clients[c].masterId].status = C!INST_STATUS_LOST
                  ∧ IF clients[c].backupId ≠ 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

$\Diamond(exec\_state \in \{\text{"success"}, \text{"fatal"}\})$

$Init \triangleq$

Initialiaze variables

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

$AtLeastOneClientStarted \triangleq$

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

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

$E\_KillingMaster \triangleq$

Kill the active master instance.

```
$\land\ exec\_state = \text{"running"}$
$\land\ AtLeastOneClientStarted$
$\land\ killed < MAX\_KILL$
$\land$ 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 $\langle exec\_state, clients, backup, msgs \rangle$
```

$E\_KillingBackup \triangleq$

Kill the active backup instance.

```
  $\land$ $exec\_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 $\langle exec\_state, clients, master, msgs \rangle$
```

$C\_Starting \triangleq$

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

```
  $\land$ $exec\_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 $\langle exec\_state, master, backup, killed \rangle$
```

$M\_Doing \triangleq$

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

```
  $\land$ $exec\_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 $\langle exec\_state, clients, backup, killed \rangle$
```

$C\_HandlingMasterDone \triangleq$

```
Client receiving “done” from master, and forwarding action to backup
$\land$ $exec\_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 $\langle exec\_state, master, backup, killed\rangle$

$B\_Doing \triangleq$
Backup receiving “do”, updating value, then sending “done”
$\land$ $exec\_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 $\langle exec\_state, clients, master, killed\rangle$

$C\_HandlingBackupDone \triangleq$
Client receiving “done” from backup. Replication completed
```

```
$\land$ $exec\_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 $![msg.clientId].phase = C!PH2\_COMPLETED]$
                 if all clients completed, then terminate the execution successfully
       $\land$ IF $\forall c \in C!CLIENT\_ID : clients'[c].phase = C!PH2\_COMPLETED$
           THEN $exec\_state' = $ “success”
           ELSE $exec\_state' = exec\_state$
$\land$ UNCHANGED $\langle master, backup, killed \rangle$
```

```
$C\_HandlingMasterDoFailed \triangleq$
  Client received the system’s notification of a dead master, and is requesting the backup to return
  the new master info
$\land$ $exec\_state = $ “running”
$\land$ LET $msg \triangleq C!FindMessageToWithTag($“m”, $C!INST\_STATUS\_LOST$, “masterDo”$)$
        $knownBackup \triangleq$ IF $msg \neq C!NOT\_MESSAGE$
                                  THEN $C!FindBackup(C!INST\_STATUS\_ACTIVE)$
                                  ELSE $C!NOT\_BACKUP$
  IN   $\land$ $msg \neq C!NOT\_MESSAGE$
       $\land$ IF $knownBackup = C!NOT\_BACKUP$
           THEN $\land$ $C!RecvMsg(msg)$
                $\land$ $exec\_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”$])$
                $\land$ $exec\_state' = exec\_state$
                $\land$ $clients' = clients$
$\land$ UNCHANGED $\langle master, backup, killed \rangle$
```

```
$C\_HandlingBackupDoFailed \triangleq$
  Client received the system’s notification of a dead backup, and is requesting the master to return
  the new backup info
$\land$ $exec\_state = $ “running”
$\land$ LET $msg \triangleq C!FindMessageToWithTag($“b”, $C!INST\_STATUS\_LOST$, “backupDo”$)$
  IN   $\land$ $msg \neq C!NOT\_MESSAGE$
       $\land$ $C!ReplaceMsg(msg, [from \mapsto$ “c”,
                                to ↦ "m",
                                clientId  ↦ msg.clientId,
                                masterId ↦ clients[msg.clientId].masterId,
                                  send the client's backup knowledge,
                                  to force the master to not respond until rereplication
                                backupId ↦ clients[msg.clientId].backupId,
                                value ↦ 0,
                                tag ↦ "masterGetNewBackup"])
  ∧ UNCHANGED ⟨exec_state, clients, master, backup, killed⟩
```

```
M_GettingNewBackup ≜
  Master responding to client with updated backup identity
  ∧ exec_state = "running"
  ∧ LET msg ≜ C!FindMessageToWithTag("m", C!INST_STATUS_ACTIVE, "masterGetNewBackup")
     IN   ∧ msg ≠ C!NOT_MESSAGE
            master must not respond until it recovers the dead backup
          ∧ msg.backupId ≠ master[msg.masterId].backupId
          ∧ C!ReplaceMsg(msg, [from ↦ "m",
                                to ↦ "c",
                                clientId  ↦ msg.clientId,
                                masterId ↦ msg.masterId,
                                backupId ↦ master[msg.masterId].backupId,
                                value ↦ 0,
                                tag ↦ "newBackupId"])
  ∧ UNCHANGED ⟨exec_state, clients, master, backup, killed⟩

B_GettingNewMaster ≜
  Backup responding to client with updated master identity
  ∧ exec_state = "running"
  ∧ LET msg ≜ C!FindMessageToWithTag("b", C!INST_STATUS_ACTIVE, "backupGetNewMaster")
     IN   ∧ msg ≠ C!NOT_MESSAGE
            backup must not respond until it recovers the dead master
          ∧ msg.masterId ≠ backup[msg.backupId].masterId
          ∧ C!ReplaceMsg(msg, [from ↦ "b",
                                to ↦ "c",
                                clientId  ↦ msg.clientId,
                                masterId ↦ backup[msg.backupId].masterId,
                                backupId ↦ msg.backupId,
                                value ↦ 0,
                                tag ↦ "newMasterId"])
  ∧ UNCHANGED ⟨exec_state, clients, master, backup, killed⟩
```

```
C_HandlingBackupGetNewMasterFailed ≜
```

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

```
$\land$ $exec\_state$ = "running"
$\land$ LET $msg \triangleq C!FindMessageToWithTag($"b", $C!INST\_STATUS\_LOST$, "backupGetNewMaster"$)$
        $foundMaster \triangleq C!FindMaster(C!INST\_STATUS\_ACTIVE)$
    IN  $\land$ $msg \neq C!NOT\_MESSAGE$
        $\land$ $C!RecvMsg(msg)$
        $\land$ IF $foundMaster = C!NOT\_MASTER$   no live master found
             THEN $\land$ $exec\_state'$ = "fatal"
                  $\land$ $clients' = [clients$ EXCEPT $![msg.clientId].phase = C!PH2\_COMPLETED\_FATAL]$
             ELSE $\land$ $exec\_state' = exec\_state$
                      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 $\langle master, backup, killed \rangle$
```

$C\_HandlingMasterGetNewBackupFailed \triangleq$

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*.

```
$\land$ $exec\_state$ = "running"
$\land$ LET $msg \triangleq C!FindMessageToWithTag($"m", $C!INST\_STATUS\_LOST$, "masterGetNewBackup"$)$
    IN  $\land$ $msg \neq C!NOT\_MESSAGE$
        $\land$ $exec\_state'$ = "fatal"
        $\land$ $clients' = [clients$ EXCEPT $![msg.clientId].phase = C!PH2\_COMPLETED\_FATAL]$
        $\land$ $C!RecvMsg(msg)$
$\land$ UNCHANGED $\langle master, backup, killed \rangle$
```

$C\_UpdatingBackupId \triangleq$

```
$\land$ $exec\_state$ = "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 $![msg.clientId].backupId = msg.backupId,$
                                         $![msg.clientId].phase = C!PH2\_COMPLETED]$
          if all clients completed, then terminate the execution successfully
        $\land$ IF $\forall c \in C!CLIENT\_ID : clients'[c].phase = C!PH2\_COMPLETED$
             THEN $exec\_state'$ = "success"
             ELSE $exec\_state' = exec\_state$
$\land$ UNCHANGED $\langle master, backup, killed \rangle$
```

```
$C\_UpdatingMasterIdAndBePending \triangleq$
  Client receiving a new master identify from a live backup and is preparing to restart by changing
  its phase to pending
  $\land\ exec\_state = \text{"running"}$
  $\land$ LET $msg \triangleq C!FindMessageToClient(\text{"b"}, \text{"newMasterId"})$
     IN  $\land\ msg \neq C!NOT\_MESSAGE$
         $\land\ C!RecvMsg(msg)$
         $\land\ clients' = [clients$ EXCEPT $![msg.clientId].masterId = msg.masterId,$
                                          $![msg.clientId].phase = C!PH1\_PENDING]$
  $\land$ UNCHANGED $\langle exec\_state, master, backup, killed \rangle$
```

```
$M\_CreatingNewBackup \triangleq$
  Master creating a new backup using its own exec_state. Master does not process any client
  requests during recovery
  $\land\ exec\_state = \text{"running"}$
  $\land$ LET $activeM \triangleq C!FindMaster(C!INST\_STATUS\_ACTIVE)$
          $activeB \triangleq C!FindBackup(C!INST\_STATUS\_ACTIVE)$
          $lostB \triangleq C!LastLostBackup$
     IN  $\land\ activeM \neq C!NOT\_MASTER$  active master exists
         $\land\ activeB = C!NOT\_BACKUP$  active backup does not exist
         $\land\ lostB \neq C!NOT\_BACKUP$  a lost backup exists
         $\land$ LET $newBackupId \triangleq lostB.id + 1$  new backup id is the following id of the dead backup
            IN  $\land\ newBackupId \leq C!MAX\_INSTANCE\_ID$
                $\land\ backup' = [backup$ EXCEPT $![newBackupId].status = C!INST\_STATUS\_ACTIVE,$
                                                $![newBackupId].masterId = activeM.id,$
                                                $![newBackupId].value = activeM.value,$
                                                $![newBackupId].version = activeM.version]$
                $\land\ master' = [master$ EXCEPT $![activeM.id].backupId = newBackupId]$
  $\land$ UNCHANGED $\langle exec\_state, clients, msgs, killed \rangle$

$B\_CreatingNewMaster \triangleq$
  Backup creating a new master using its own exec_state. Backup does not process any client
  requests during recovery
  $\land\ exec\_state = \text{"running"}$
  $\land$ LET $activeM \triangleq C!FindMaster(C!INST\_STATUS\_ACTIVE)$
          $activeB \triangleq C!FindBackup(C!INST\_STATUS\_ACTIVE)$
          $lostM \triangleq C!LastLostMaster$
     IN  $\land\ activeM = C!NOT\_MASTER$  active master does not exist
         $\land\ activeB \neq C!NOT\_BACKUP$  active backup exists
         $\land\ lostM \neq C!NOT\_MASTER$  a lost master exists
         $\land$ LET $newMasterId \triangleq lostM.id + 1$
            IN  $\land\ newMasterId \leq C!MAX\_INSTANCE\_ID$
                $\land\ master' = [master$ EXCEPT $![newMasterId].status = C!INST\_STATUS\_ACTIVE,$
                                                $![newMasterId].backupId = activeB.id,$
                                                $![newMasterId].value = activeB.value,$
```

$![newMasterId].version = activeB.version]$

$\land\ backup' = [backup\ \text{EXCEPT}\ ![activeB.id].masterId = newMasterId\ ]$

$\land\ \text{UNCHANGED}\ \langle exec\_state,\ clients,\ msgs,\ killed \rangle$

$Next\ \triangleq$

$\lor\ E\_KillingMaster$
$\lor\ E\_KillingBackup$
$\lor\ C\_Starting$
$\lor\ M\_Doing$
$\lor\ C\_HandlingMasterDone$
$\lor\ B\_Doing$
$\lor\ C\_HandlingBackupDone$
$\lor\ C\_HandlingMasterDoFailed$
$\lor\ C\_HandlingBackupDoFailed$
$\lor\ M\_GettingNewBackup$
$\lor\ B\_GettingNewMaster$
$\lor\ C\_HandlingBackupGetNewMasterFailed$
$\lor\ C\_HandlingMasterGetNewBackupFailed$
$\lor\ C\_UpdatingBackupId$
$\lor\ C\_UpdatingMasterIdAndBePending$
$\lor\ M\_CreatingNewBackup$
$\lor\ B\_CreatingNewMaster$

$Liveness\ \triangleq$

$\land\ \text{WF}_{Vars}(E\_KillingMaster)$
$\land\ \text{WF}_{Vars}(E\_KillingBackup)$
$\land\ \text{WF}_{Vars}(C\_Starting)$
$\land\ \text{WF}_{Vars}(M\_Doing)$
$\land\ \text{WF}_{Vars}(C\_HandlingMasterDone)$
$\land\ \text{WF}_{Vars}(B\_Doing)$
$\land\ \text{WF}_{Vars}(C\_HandlingBackupDone)$
$\land\ \text{WF}_{Vars}(C\_HandlingMasterDoFailed)$
$\land\ \text{WF}_{Vars}(C\_HandlingBackupDoFailed)$
$\land\ \text{WF}_{Vars}(M\_GettingNewBackup)$
$\land\ \text{WF}_{Vars}(B\_GettingNewMaster)$
$\land\ \text{WF}_{Vars}(C\_HandlingBackupGetNewMasterFailed)$
$\land\ \text{WF}_{Vars}(C\_HandlingMasterGetNewBackupFailed)$
$\land\ \text{WF}_{Vars}(C\_UpdatingBackupId)$
$\land\ \text{WF}_{Vars}(C\_UpdatingMasterIdAndBePending)$
$\land\ \text{WF}_{Vars}(M\_CreatingNewBackup)$
$\land\ \text{WF}_{Vars}(B\_CreatingNewMaster)$

Specification

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

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