MODULE *ExecutorDistFinishCorrectRep*

This specification models a subset of $X10$ programs to verify the correctness of the 'finish' construct, which provides a termination detection protocol.

Distributed *Finish*:

This module specifies a distributed finish implementation that replicates the finish state on two places to allow correct termination when one replica is lost

Fixing $PPoPP14$ Replication Bug:

We corrected a replication bug that was found in the original distributed finish implementation, that was published in $PPoPP14$.

$PPoPP14$ wrong replication:

Normal path: requester $\rightarrow$ master *do*();
master $\rightarrow$ backup *do*();
backup $\rightarrow$ master return;
master $\rightarrow$ requester return;
If Master died: *requestor* $\rightarrow$ backup *do*(); or *requestor* $\rightarrow$ *adopter* *do*(); if backup was adopted.
Error: the action *do*(); may be performed twice on the backup.

Corrected replication:

Normal path: *requestor* $\rightarrow$ master *do*();
master $\rightarrow$ *requestor* return;
requestor $\rightarrow$ backup *do*();
backup $\rightarrow$ *requestor* return;
If Master died: *requestor* $\rightarrow$ backup *getAdopter*();
requestor $\rightarrow$ *adopter* *do*();
The action *do*(); will be performed once in all cases

EXTENDS *Integers*, *Sequences*, *TLC*

Constants

CONSTANTS

PLACE,	The set of places
PROG_HOME,	The home place from which the program starts
PROG,	The input program
MXFINISHES,	Maximum finish objects including root and remote
BACKUP,	A function from place to its backup
DEPTH	Maximum expected depth of the trace

Variables

VARIABLES

fstates,	Array of finish states
fmasters,	Master finish states
fbackups,	Backup finish states

msgs,	The set of inflight messages. We delete a message once received
pstate,	Program state: *init* → *running* → *terminated*
seq,	Sequences
thrds,	Threads at all places
killed,	The set places killed so far
pendingAct,	Set of activities received at destination place but need permission from the resilient store to run
runningThrds,	Set of running threads in all places
blockedThrds,	Set of blocked threads in all places
waitForMsgs,	Messages that blocked threads are waiting for. If the sender dies, we send them with a failed status to unblock these threads
mastersStatus,	The status of the master stores at each place
adoptSet,	Recovery variable: set of finishes that need adoption
convertSet,	Recovery variable: steps to convert dead tasks to 0*s*
actionName,	Debugging variable: the current action name
depth	Debugging variable: the current depth

$$Vars \triangleq \langle fstates,\ msgs,\ pstate,\ seq,\ thrds,\ killed,\ pendingAct,\ fmasters,\ fbackups,\ waitForMsgs,\ mastersStatus,\ adoptSet,\ convertSet,\ blockedThrds,\ runningThrds,\ actionName,\ depth\rangle$$

Predicate to hide the finish implementation

$$Finish(fid) \triangleq \text{INSTANCE } DistFinish$$

$$C \triangleq \text{INSTANCE } Commons$$

$$GetRootFinishId(fid) \triangleq$$
$$\text{IF } fid = C!NoParent \text{ THEN } C!NotID$$
$$\text{ELSE IF } Finish(fid)!IsRoot \text{ THEN } fid$$
$$\text{ELSE } fstates[fid].root$$

Invariants (formulas true in every reachable state.)

$$TypeOK \triangleq$$
$$\land\ fstates \in [C!IDRange \to\ C!FinishState]$$
$$\land\ thrds \in\ [PLACE \to [\ C!ThreadID \to\ C!Thread]]$$
$$\land\ msgs \subseteq\ C!Messages$$
$$\land\ pstate \in \{\text{“running”}, \text{“terminated”}\}$$
$$\land\ PROG \in [C!BlockID \to\ C!Block]$$
$$\land\ PROG\_HOME \in PLACE$$
$$\land\ seq \in C!Sequences$$
$$\land\ killed \subseteq PLACE$$

$\land$ $pendingAct \subseteq C!Activity$
$\land$ $fmasters \in [C!IDRange \to C!MasterFinish]$
$\land$ $fbackups \in [C!IDRange \to C!BackupFinish]$
$\land$ $BACKUP \in [PLACE \to PLACE]$
$\land$ $mastersStatus \in [PLACE \to C!MasterStatus]$
$\land$ $adoptSet \subseteq C!Adopter$
$\land$ $convertSet \subseteq C!ConvTask$
$\land$ $runningThrds \subseteq C!PlaceThread$
$\land$ $blockedThrds \subseteq C!PlaceThread$
$\land$ $depth \in 0 \,..\, DEPTH + 1$

$StateOK \triangleq$ TRUE

$MustTerminate \triangleq$
$\Diamond(pstate =$ "terminated")

Initialization

$Init \triangleq$
$\land$ $actionName = \langle$"Init", $PROG\_HOME\rangle$
$\land$ $depth = 0$
$\land$ $fstates = [r \in C!IDRange \mapsto$
$[id \mapsto C!NotID,\ status \mapsto$ "unused", $type \mapsto$ "NA",
$count \mapsto 0,\ here \mapsto C!NotPlace,$
$parent \mapsto C!NotID,\ root \mapsto C!NotID,\ isGlobal \mapsto$ FALSE,
$eroot \mapsto C!NotID]]$
$\land$ $fmasters = [r \in C!IDRange \mapsto$
$[\ id \mapsto C!NotID,$
$numActive \mapsto 0,$
$live \mapsto [p \in PLACE \mapsto 0],$
$transit \mapsto [p \in PLACE \mapsto [q \in PLACE \mapsto 0]],$
$liveAdopted \mapsto [p \in PLACE \mapsto 0],$
$transitAdopted \mapsto [p \in PLACE \mapsto [q \in PLACE \mapsto 0]],$
$children \mapsto \{\},$
$backupPlace \mapsto C!NotPlace\ ,$
$isReleased \mapsto$ FALSE]]
$\land$ $fbackups = [r \in C!IDRange \mapsto$
$[\ id \mapsto C!NotID,$
$live \mapsto [p \in PLACE \mapsto 0],$
$transit \mapsto [p \in PLACE \mapsto [q \in PLACE \mapsto 0]],$
$children \mapsto \{\},$
$isAdopted \mapsto$ FALSE,
$adoptedRoot \mapsto C!NotID,$
$numActive \mapsto 0,$
$isReleased \mapsto$ FALSE]]
$\land$ $pstate =$ "running"

$\land$ *mastersStatus* $= [p \in PLACE \mapsto [$ *status* $\mapsto$ "running",
lastKilled $\mapsto C!NotPlace]]$
$\land$ *msgs* $= \{\}$
$\land$ *seq* $= [aseq \mapsto 1, fseq \mapsto C!FIRST\_ID, mseq \mapsto 1]$
$\land$ *thrds* $= [p \in PLACE \mapsto$ start with one running thread at *PROG_HOME*
$[t \in C!ThreadID \mapsto$
IF $p = PROG\_HOME \land t = 0$
THEN $[tid \mapsto t,$ *status* $\mapsto$ "running",
blockingType $\mapsto$ "NA",
stack $\mapsto \langle[b \mapsto 0,$
$i \mapsto$ IF $PROG[0].type =$ "finish"
THEN $C!I\_PRE\_FIN\_ALLOC$
ELSE $C!I\_START,$
fid $\mapsto C!NoParent]\rangle]$
ELSE $[tid \mapsto t,$ *status* $\mapsto$ "idle",
blockingType $\mapsto$ "NA",
stack $\mapsto \langle\rangle]]]$
$\land$ *runningThrds* $= \{[here \mapsto PROG\_HOME, tid \mapsto 0]\}$
$\land$ *blockedThrds* $= \{\}$
$\land$ *killed* $= \{\}$
$\land$ *pendingAct* $= \{\}$
$\land$ *waitForMsgs* $= \{\}$
$\land$ *adoptSet* $= \{\}$
$\land$ *convertSet* $= \{\}$

Helper Actions

SetActionNameAndDepth(*name*) $\triangleq$
IF *depth* $=$ *DEPTH* THEN TRUE ELSE $\land$ *actionName*$' =$ *name* $\land$ *depth*$' =$ *depth* $+ 1$

FindPendingActivity(*actId*) $\triangleq$
LET *aset* $\triangleq \{a \in pendingAct : a.aid = actId\}$
IN IF *aset* $= \{\}$ THEN $C!NotActivity$
ELSE CHOOSE $x \in aset$: TRUE

FindIdleThread(*here*) $\triangleq$
LET *idleThreads* $\triangleq C!PlaceThread \setminus (runningThrds \cup blockedThrds)$
tset $\triangleq \{t \in idleThreads :$
$\land\ t.here = here$
$\land\ t.here \notin killed$
$\land\ thrds[t.here][t.tid].status =$ "idle" $\}$
IN IF *tset* $= \{\}$ THEN $C!NotPlaceThread$
ELSE CHOOSE $x \in tset$: TRUE

Program Execution Actions

```
FindRunningThreadForStartFinish ≜
  LET tset ≜ {t ∈ runningThrds :
                ∧ t.here ∉ killed
                ∧ thrds[t.here][t.tid].status = "running"
                ∧ LET top ≜ Head(thrds[t.here][t.tid].stack)
                      blk ≜ top.b
                      lstStmt ≜ top.i
                  IN  ∧ PROG[blk].type = "finish"
                      ∧ lstStmt = C!I_PRE_FIN_ALLOC}
  IN  IF tset = {} THEN C!NotPlaceThread
      ELSE CHOOSE x ∈ tset : TRUE

Running thread processing the beginning of a finish block
StartFinish ≜
  ∧ pstate = "running"
  ∧ LET pthrd ≜ FindRunningThreadForStartFinish
    IN  ∧ pthrd ≠ C!NotPlaceThread
        ∧ LET here ≜ pthrd.here
              tid ≜ pthrd.tid
              top ≜ Head(thrds[here][tid].stack)
              tail ≜ Tail(thrds[here][tid].stack)
              lstStmt ≜ top.i
              curStmt ≜ top.i + 1
              blk ≜ top.b
              fid ≜ top.fid
              newFid ≜ seq.fseq
              encRoot ≜ C!GetEnclosingRoot(fid, newFid)
          IN  ∧ SetActionNameAndDepth(⟨"StartFinish", here⟩)
              ∧ Finish(seq.fseq)!Alloc(C!ROOT_FINISH, here, fid, newFid)
              ∧ C!IncrFSEQ
              ∧ thrds' = [thrds EXCEPT ![here][tid].stack =
                                        ⟨[b   ↦ top.b,
                                          i   ↦ curStmt,
                                          fid ↦ seq.fseq]
                                        ⟩ ∘ tail]
              ∧ IF seq.fseq = C!FIRST_ID
                THEN ∧ fmasters' = fmasters   will be initialized in transit
                     ∧ fbackups' = fbackups
                ELSE ∧ fmasters' = [fmasters EXCEPT ![encRoot].children =
                                                    @ ∪ {newFid}]
                     ∧ fbackups' = [fbackups EXCEPT ![encRoot].children =
                                                    @ ∪ {newFid}]
  ∧ UNCHANGED ⟨convertSet, adoptSet, mastersStatus, pstate, killed, pendingAct,
               msgs, waitForMsgs, runningThrds, blockedThrds⟩
```

```
FindRunningThreadForScheduleNestedFinish ≜
  LET tset ≜ {t ∈ runningThrds :
                ∧ t.here ∉ killed
                ∧ thrds[t.here][t.tid].status = "running"
                ∧ LET top ≜ Head(thrds[t.here][t.tid].stack)
                      blk ≜ top.b
                      curStmt ≜ top.i + 1
                       nested ≜ PROG[blk].stmts[curStmt]
                  IN   ∧ PROG[blk].type ∉ {"expr", "kill"}
                       ∧ curStmt ≥ 0
                       ∧ curStmt ≤ PROG[blk].mxstmt
                       ∧ PROG[nested].type = "finish"
                       ∧ PROG[nested].dst = t.here }
  IN  IF tset = {} THEN C!NotPlaceThread
      ELSE CHOOSE x ∈ tset : TRUE

Processing a nested finish in the currently running block
ScheduleNestedFinish ≜
  ∧ pstate = "running"
  ∧ LET pthrd ≜ FindRunningThreadForScheduleNestedFinish
    IN   ∧ pthrd ≠ C!NotPlaceThread
         ∧ LET here ≜ pthrd.here
                tid ≜ pthrd.tid
                top ≜ Head(thrds[here][tid].stack)
                tail ≜ Tail(thrds[here][tid].stack)
                lstStmt ≜ top.i
                curStmt ≜ top.i + 1
                blk ≜ top.b
                fid ≜ top.fid
                nested ≜ PROG[blk].stmts[curStmt]
                newFid ≜ seq.fseq
                 encRoot ≜ C!GetEnclosingRoot(fid, newFid)
           IN   ∧ SetActionNameAndDepth(⟨"ScheduleNestedFinish", here⟩)
                ∧ thrds' = [thrds EXCEPT ![here][tid].stack =
                                          ⟨[ b ↦ nested,
                                             i ↦ C!I_START,
                                           fid ↦ newFid],
                                           [ b ↦ top.b,
                                             i ↦ curStmt,
                                           fid ↦ fid]
                                          ⟩ ∘ tail]
                ∧ Finish(seq.fseq)!Alloc(C!ROOT_FINISH, here, fid, newFid)
                ∧ C!IncrFSEQ
                ∧ fmasters' = [fmasters EXCEPT ![encRoot].children =
```

$$@ \cup \{newFid\}]$$
$$\land\ fbackups' = [fbackups \text{ EXCEPT } ![encRoot].children = @ \cup \{newFid\}]$$
$$\land \text{ UNCHANGED } \langle convertSet,\ adoptSet,\ mastersStatus,\ msgs,\ pstate,\ waitForMsgs,\ killed,\ pendingAct,\ runningThrds,\ blockedThrds\rangle$$

$FindRunningThreadForSpawnLocalAsync \triangleq$
LET $tset \triangleq \{t \in runningThrds :$
$\land\ t.here \notin killed$
$\land\ thrds[t.here][t.tid].status = \text{“running”}$
$\land$ LET $top \triangleq Head(thrds[t.here][t.tid].stack)$
$blk \triangleq top.b$
$curStmt \triangleq top.i + 1$
$nested \triangleq PROG[blk].stmts[curStmt]$
IN $\land\ PROG[blk].type \notin \{\text{“expr”},\ \text{“kill”}\}$
$\land\ curStmt \geq 0$
$\land\ curStmt \leq PROG[blk].mxstmt$
$\land\ PROG[nested].type = \text{“async”}$
$\land\ PROG[nested].dst = t.here$
$\}$
IN IF $tset = \{\}$ THEN $C!NotPlaceThread$
ELSE CHOOSE $x \in tset$: TRUE

Processing a nested local *async* in the currently running block
$SpawnLocalAsync \triangleq$
$\land\ pstate = \text{“running”}$
$\land$ LET $pthrd \triangleq FindRunningThreadForSpawnLocalAsync$
IN $\land\ pthrd \neq C!NotPlaceThread$
$\land$ LET $here \triangleq pthrd.here$
$tid \triangleq pthrd.tid$
$top \triangleq Head(thrds[here][tid].stack)$
$tail \triangleq Tail(thrds[here][tid].stack)$
$lstStmt \triangleq top.i$
$curStmt \triangleq top.i + 1$
$blk \triangleq top.b$
$fid \triangleq top.fid$
$nested \triangleq PROG[blk].stmts[curStmt]$
$idle \triangleq FindIdleThread(here)$
$act \triangleq [aid \mapsto seq.aseq,\ b \mapsto nested,\ fid \mapsto fid]$
$stkEntry \triangleq [b \mapsto act.b,\ i \mapsto C!I\_START,\ fid \mapsto act.fid]$
IN $\land\ SetActionNameAndDepth(\langle\text{“SpawnLocalAsync”},\ here\rangle)$
$\land$ IF $act.fid \neq C!NoParent$
THEN $Finish(act.fid)!NotifyLocalActivitySpawnAndCreation(here,\ act)$
ELSE $fstates' = fstates$

$\land\ C!IncrASEQ$
$\land\ thrds' = [thrds \text{ EXCEPT } ![here][tid].stack =$
$\langle[\ b\ \mapsto top.b,$
$i\ \mapsto curStmt,$
$fid \mapsto fid]$
$\rangle \circ tail,$
$![here][idle.tid].stack = \langle stkEntry \rangle,$
$![here][idle.tid].status = \text{“running”}]$
$\land\ runningThrds' = runningThrds \cup \{[here \mapsto here,\ tid \mapsto idle.tid]\}$
$\land$ UNCHANGED $\langle convertSet,\ adoptSet,\ mastersStatus,\ msgs,\ pstate,\ killed,$
$pendingAct,\ fmasters,\ fbackups,\ waitForMsgs,\ blockedThrds\rangle$

$FindRunningThreadForSpawnRemoteAsync \triangleq$
LET $tset \triangleq \{t \in runningThrds :$
$\land\ t.here \notin killed$
$\land\ thrds[t.here][t.tid].status = \text{“running”}$
$\land$ LET $top \triangleq Head(thrds[t.here][t.tid].stack)$
$fid \triangleq top.fid$
$blk \triangleq top.b$
$curStmt \triangleq top.i + 1$
$nested \triangleq PROG[blk].stmts[curStmt]$
IN $\land\ PROG[blk].type \notin \{\text{“expr”},\ \text{“kill”}\}$
$\land\ fid \neq C!NoParent$
$\land\ curStmt \geq 0$
$\land\ curStmt \leq PROG[blk].mxstmt$
$\land\ PROG[nested].type = \text{“async”}$
$\land\ PROG[nested].dst \neq t.here$
$\}$
IN IF $tset = \{\}$ THEN $C!NotPlaceThread$
ELSE CHOOSE $x \in tset$: TRUE

Processing a nested remote *async* in the currently running block
$SpawnRemoteAsync \triangleq$
$\land\ pstate = \text{“running”}$
$\land$ LET $pthrd \triangleq FindRunningThreadForSpawnRemoteAsync$
IN $\land\ pthrd \neq C!NotPlaceThread$
$\land$ LET $here \triangleq pthrd.here$
$tid \triangleq pthrd.tid$
$top \triangleq Head(thrds[here][tid].stack)$
$tail \triangleq Tail(thrds[here][tid].stack)$
$lstStmt \triangleq top.i$
$curStmt \triangleq top.i + 1$
$blk \triangleq top.b$
$fid \triangleq top.fid$

```
                root ≜ GetRootFinishId(fid)
                nested ≜ PROG[blk].stmts[curStmt]
                dst ≜ PROG[nested].dst
          IN    ∧ SetActionNameAndDepth(⟨"SpawnRemoteAsync", here, "to", dst⟩)
                ∧ Finish(fid)!NotifySubActivitySpawn(dst)
                ∧ thrds' = [thrds EXCEPT ![here][tid].status = "blocked",
                                         ![here][tid].blockingType = "AsyncTransit"]
                ∧ blockedThrds' = blockedThrds ∪ {[here ↦ here, tid ↦ tid]}
                ∧ runningThrds' = runningThrds \ {[here ↦ here, tid ↦ tid]}
∧ UNCHANGED ⟨convertSet, adoptSet, mastersStatus, pstate, killed, pendingAct,
             fmasters, fbackups⟩

──────────────────────────────────────────────────────────────────────────────

FindRunningThreadForRunExprOrKill ≜
  LET tset ≜ {t ∈ runningThrds :
                ∧ t.here ∉ killed
                ∧ thrds[t.here][t.tid].status = "running"
                ∧ LET top ≜ Head(thrds[t.here][t.tid].stack)
                      blk ≜ top.b
                      curStmt ≜ top.i + 1
                       nested ≜ PROG[blk].stmts[curStmt]
                   IN   ∧ PROG[blk].type ∉ {"expr", "kill"}
                        ∧ curStmt ≥ 0
                        ∧ curStmt ≤ PROG[blk].mxstmt
                        ∧ PROG[nested].type ∈ {"expr", "kill"} }
  IN  IF tset = {} THEN C!NotPlaceThread
      ELSE CHOOSE x ∈ tset : TRUE

Kill(dead) ≜
  ∧ killed' = killed ∪ {dead}
  ∧ adoptSet' = adoptSet ∪ {m ∈ C!Adopter :
                              ∧ m.child ≠ C!NotID
                              ∧ m.adopter ≠ C!NotID
                              ∧ m.here ≠ dead
                              ∧ m.here = fstates[m.adopter].here
                              ∧ m.child ∈ fmasters[m.adopter].children
                              ∧ fbackups[m.child].isAdopted = FALSE
                              ∧ fstates[m.child].here = dead
                              ∧ m.adopter = fstates[m.child].eroot}
  ∧ IF adoptSet' = {}
    THEN ∧ mastersStatus' = [mastersStatus EXCEPT ![PROG_HOME].status = "convertDead",
                                                  ![PROG_HOME].lastKilled = dead]
    ELSE ∧ mastersStatus' = [p ∈ PLACE ↦ IF ∃ m ∈ adoptSet' : m.here = p
                                         THEN [ status ↦ "seekAdoption",
                                                lastKilled ↦ dead]
```

```
                                                ELSE [     status   ↦ "running",
                                                      lastKilled ↦ C!NotPlace]]
  ∧ convertSet' = {t ∈ C!ConvTask :
                     ∧ t.pl ≠ C!NotPlace
                     ∧ t.pl ≠ dead
                     ∧ t.pl ∉ killed
                     ∧ t.fid ∈ {id ∈ C!IDRange :
                                  ∧ fmasters[id].id ≠ C!NotID
                                  ∧ fstates[id].here ≠ dead}
                     ∧ t.here = fstates[t.fid].here}
  ∧ LET delMsgs ≜ {m ∈ msgs : m.dst = dead }      delete messages going to a dead place
        wfm ≜ {m ∈ waitForMsgs : m.dst = dead}    delete waitForMsgs to a dead place
    IN  ∧ msgs' = msgs \ delMsgs
        ∧ waitForMsgs' = waitForMsgs \ wfm

Processing a nested expression in the currently running block
RunExprOrKill ≜
  ∧ pstate = "running"
  ∧ LET pthrd ≜ FindRunningThreadForRunExprOrKill
    IN  ∧ pthrd ≠ C!NotPlaceThread
        ∧ LET here ≜ pthrd.here
              tid ≜ pthrd.tid
              top ≜ Head(thrds[here][tid].stack)
              tail ≜ Tail(thrds[here][tid].stack)
              lstStmt ≜ top.i
              curStmt ≜ top.i + 1
              blk ≜ top.b
              fid ≜ top.fid
              nested ≜ PROG[blk].stmts[curStmt]
          IN  ∧ SetActionNameAndDepth(⟨"RunExprOrKill", here, PROG[nested].type⟩)
              ∧ thrds' = [thrds EXCEPT ![here][tid].stack =
                                          ⟨[ b   ↦ top.b,
                                             i   ↦ curStmt,
                                             fid ↦ fid]
                                          ⟩ ∘ tail]
              ∧ IF PROG[nested].type = "expr"
                THEN ∧ killed' = killed
                     ∧ PROG[nested].dst = here
                     ∧ adoptSet' = adoptSet
                     ∧ mastersStatus' = mastersStatus
                     ∧ convertSet' = convertSet
                     ∧ msgs' = msgs
                     ∧ waitForMsgs' = waitForMsgs
                ELSE ∧ Kill(PROG[nested].dst)
  ∧ UNCHANGED ⟨fstates, pstate, seq, pendingAct, fmasters, fbackups,
```

```
                    runningThrds, blockedThrds⟩

FindRunningThreadForTerminateAsync ≜
  LET tset ≜ {t ∈ runningThrds :
                ∧ t.here ∉ killed
                ∧ thrds[t.here][t.tid].status = "running"
                ∧ LET top ≜ Head(thrds[t.here][t.tid].stack)
                      blk ≜ top.b
                      fid ≜ top.fid
                  IN  ∧ PROG[blk].type = "async"
                      ∧ PROG[blk].mxstmt = top.i }
  IN  IF tset = {} THEN C!NotPlaceThread
      ELSE CHOOSE x ∈ tset : TRUE

Running thread processing the end of an async block
TerminateAsync ≜
  ∧ pstate = "running"
  ∧ LET pthrd ≜ FindRunningThreadForTerminateAsync
    IN  ∧ pthrd ≠ C!NotPlaceThread
        ∧ LET here ≜ pthrd.here
              tid ≜ pthrd.tid
              top ≜ Head(thrds[here][tid].stack)
              blk ≜ top.b
              fid ≜ top.fid
          IN  ∧ SetActionNameAndDepth(⟨"TerminateAsync", here⟩)
              ∧ Finish(fid)!NotifyActivityTermination(FALSE)
              ∧ thrds' = [thrds EXCEPT ![here][tid].status = "blocked",
                                       ![here][tid].blockingType = "AsyncTerm"]
              ∧ runningThrds' = runningThrds \ {[here ↦ here, tid ↦ tid]}
              ∧ blockedThrds' = blockedThrds ∪ {[here ↦ here, tid ↦ tid]}
  ∧ UNCHANGED ⟨convertSet, adoptSet, mastersStatus, pstate, killed,
               pendingAct, fmasters, fbackups⟩

─────────────────────────────────────────────────────────────────────

FindRunningThreadForStopFinish ≜
  LET tset ≜ {t ∈ runningThrds :
                ∧ t.here ∉ killed
                ∧ thrds[t.here][t.tid].status = "running"
                ∧ LET top ≜ Head(thrds[t.here][t.tid].stack)
                  IN  ∧ PROG[top.b].type = "finish"
                      ∧ PROG[top.b].mxstmt = top.i }
  IN  IF tset = {} THEN C!NotPlaceThread
      ELSE CHOOSE x ∈ tset : TRUE

Running thread processing the end of a finish block and blocking itself
StopFinish ≜
```

$\land\ pstate = \text{“running”}$
$\land$ LET $pthrd \triangleq FindRunningThreadForStopFinish$
IN $\land\ pthrd \neq C!NotPlaceThread$
$\land$ LET $here \triangleq pthrd.here$
$tid \triangleq pthrd.tid$
$top \triangleq Head(thrds[here][tid].stack)$
IN $\land\ SetActionNameAndDepth(\langle \text{“StopFinish”}, here\rangle)$
$\land\ PROG[top.b].type = \text{“finish”}$
$\land\ PROG[top.b].mxstmt = top.i$
$\land\ Finish(top.fid)!NotifyActivityTermination(\text{TRUE})$
$\land\ thrds' = [thrds \text{ EXCEPT } ![here][tid].status = \text{“blocked”},$
$![here][tid].blockingType = \text{“FinishEnd”}]$
$\land\ runningThrds' = runningThrds \setminus \{[here \mapsto here,\ tid \mapsto tid]\}$
$\land\ blockedThrds' = blockedThrds \cup \{[here \mapsto here,\ tid \mapsto tid]\}$
$\land$ UNCHANGED $\langle convertSet,\ adoptSet,\ mastersStatus,\ pstate,\ killed,\ pendingAct,$
$fmasters,\ fbackups\rangle$

$RecvAsync \triangleq$
$\land\ pstate = \text{“running”}$
$\land$ LET $msg \triangleq C!FindMSG(\text{“async”})$
IN $\land\ msg \neq C!NotMessage$
$\land$ LET $here \triangleq msg.dst$
$pid \triangleq msg.fid$
$fid \triangleq C!GetActiveFID(C!REMOTE\_FINISH,\ here,\ pid)$
$src \triangleq msg.src$
$blk \triangleq msg.b$
$newFID \triangleq seq.fseq$
$activity \triangleq [aid \mapsto seq.aseq,\ b \mapsto blk,\ fid \mapsto newFID]$
IN $\land\ SetActionNameAndDepth(\langle \text{“RecvAsync”}, here\rangle)$
$\land\ pid \neq C!NotID$
$\land\ fid = C!NotID$ we don't reuse remote finishes
$\land\ src \neq C!NotPlace$
$\land\ Finish(activity.fid)!AllocRemoteAndNotifyRemoteActivityCreation($
$src,\ activity,\ msg,\ C!REMOTE\_FINISH,$
$here,$ parent$pid,$ root$pid)$
$\land\ pendingAct' = pendingAct \cup \{activity\}$
$\land\ C!IncrAll$
$\land$ UNCHANGED $\langle convertSet,\ adoptSet,\ mastersStatus,\ pstate,\ thrds,$
$killed,\ fmasters,\ fbackups,\ blockedThrds,\ runningThrds\rangle$

$FindBlockedThreadMasterTransitDone \triangleq$
LET $tset \triangleq \{t \in blockedThrds :$
$\land\ t.here \notin killed$

$\wedge\ thrds[t.here][t.tid].status = \text{“blocked”}$
$\wedge\ thrds[t.here][t.tid].blockingType = \text{“AsyncTransit”}$
$\wedge\ C!FindIncomingMSG(t.here,\ \text{“masterTransitDone”}) \neq C!NotMessage\ \}$
IN IF $tset = \{\}$ THEN $C!NotPlaceThread$
ELSE CHOOSE $x \in tset$: TRUE

$MasterTransitDone\ \triangleq$
$\wedge\ pstate = \text{“running”}$
$\wedge\ msgs \neq \{\}$
$\wedge$ LET $pthrd\ \triangleq\ FindBlockedThreadMasterTransitDone$
IN $\wedge\ pthrd \neq C!NotPlaceThread$
$\wedge$ LET $here\ \triangleq\ pthrd.here$
$tid\ \triangleq\ pthrd.tid$
$msg\ \triangleq\ C!FindIncomingMSG(here,\ \text{“masterTransitDone”})$
$success\ \triangleq\ msg.success$
$submit\ \triangleq\ msg.submit$
$top\ \triangleq\ Head(thrds[here][tid].stack)$
$tail\ \triangleq\ Tail(thrds[here][tid].stack)$
$lstStmt\ \triangleq\ top.i$
$curStmt\ \triangleq\ top.i + 1$
$blk\ \triangleq\ top.b$
$root\ \triangleq\ msg.fid$
$fid\ \triangleq\ top.fid$
$rootPlace\ \triangleq\ C!GetFinishHome(root)$
$nested\ \triangleq\ PROG[blk].stmts[curStmt]$
$asyncDst\ \triangleq\ PROG[nested].dst$
$isAdopter\ \triangleq\ msg.isAdopter$
$backupPlace\ \triangleq\ msg.backupPlace$
$adoptedFID\ \triangleq\ msg.adoptedFID$
$masterWFM\ \triangleq\ [src \mapsto rootPlace,$
$dst \mapsto here,$
$fid \mapsto root,$
$target \mapsto asyncDst,$
$type \mapsto \text{“masterTransitDone”}\ ]$
$backupWFM\ \triangleq\ [src \mapsto backupPlace,$
$dst \mapsto here,$
$fid \mapsto root,$
$target \mapsto asyncDst,$
$isAdopter \mapsto isAdopter,$
$adoptedFID \mapsto adoptedFID,$
$type \mapsto \text{“backupTransitDone”}\ ]$
IN $\wedge\ SetActionNameAndDepth(\langle \text{“MasterTransitDone”},\ here,$
$\text{“success”},\ success,$
$\text{“submit”},\ submit\rangle)$
Technically, we should check the condition $rootPlace \notin killed$

if success is true. we should communicate with the backup normally.
the backup then should reject the request and notify the requester
that the master has changed, so that we redirect the call to the
new master.

$\land$ IF $success \land submit \land rootPlace \notin killed$
THEN $\land\ C!ReplaceMsg(msg, [\ mid \mapsto seq.mseq,\ src \mapsto here,\ dst \mapsto backupPlace,\ target \mapsto asyncDst,\ fid \mapsto root,\ isAdopter \mapsto isAdopter,\ adoptedFID \mapsto adoptedFID,\ type \mapsto$ "backupTransit"$])$
$\land\ thrds' = thrds$
$\land\ blockedThrds' = blockedThrds$
$\land\ runningThrds' = runningThrds$
$\land\ waitForMsgs' = (waitForMsgs \setminus \{masterWFM\}) \cup \{backupWFM\}$
$\land\ C!IncrMSEQ(1)$
ELSE IF $success \land rootPlace \notin killed$ ignore the *async*, go to the next step
THEN $\land\ C!RecvMsg(msg)$
$\land\ thrds' = [thrds$ EXCEPT $![here][tid].status =$ "running",
$![here][tid].stack = \langle[\ b \mapsto top.b,\ i \mapsto curStmt,\ fid \mapsto fid] \rangle \circ tail]$
$\land\ blockedThrds' = blockedThrds \setminus \{[here \mapsto here,\ tid \mapsto tid]\}$
$\land\ runningThrds' = runningThrds \cup \{[here \mapsto here,\ tid \mapsto tid]\}$
$\land\ waitForMsgs' = waitForMsgs \setminus \{masterWFM\}$
$\land\ C!IncrMSEQ(1)$
ELSE $\land\ C!ReplaceMsg(msg, [\ mid \mapsto seq.mseq,\ src \mapsto here,\ dst \mapsto C!GetBackup(rootPlace),\ source \mapsto here,\ target \mapsto asyncDst,\ fid \mapsto root,\ type \mapsto$ "backupGetAdopter", $actionType \mapsto$ "transit", $aid \mapsto C!NotActivity.aid,\ finishEnd \mapsto$ FALSE$])$
$\land\ thrds' = thrds$
$\land\ blockedThrds' = blockedThrds$
$\land\ runningThrds' = runningThrds$
$\land\ waitForMsgs' = waitForMsgs \setminus \{masterWFM\}$
we don't expect the backup to die

that is why we don't add
backupGetAdopterDone in *waitForMsgs*

$\wedge\ C!IncrMSEQ(1)$

$\wedge$ UNCHANGED $\langle convertSet,\ adoptSet,\ mastersStatus,\ fstates,\ pstate,\ killed,\ pendingAct,\ fmasters,\ fbackups\rangle$

$MasterLiveDone \triangleq$
$\wedge\ pstate = \text{"running"}$
$\wedge\ pendingAct \neq \{\}$
$\wedge\ msgs \neq \{\}$
$\wedge$ LET $msg \triangleq C!FindMSG(\text{"masterLiveDone"})$
IN $\wedge\ msg \neq C!NotMessage$
$\wedge$ LET $here \triangleq msg.dst$
$actId \triangleq msg.aid$
$activity \triangleq FindPendingActivity(actId)$
$root \triangleq msg.fid$
$submit \triangleq msg.submit$
$success \triangleq msg.success$
$rootPlace \triangleq C!GetFinishHome(root)$
$isAdopter \triangleq msg.isAdopter$
$adoptedFID \triangleq msg.adoptedFID$
$backupPlace \triangleq msg.backupPlace$
$source \triangleq msg.source$
$target \triangleq msg.target$
$masterWFM \triangleq [src \mapsto rootPlace,\ dst \mapsto here,\ fid \mapsto root,\ aid \mapsto actId,\ source \mapsto source,\ target \mapsto target,\ type \mapsto \text{"masterLiveDone"}\ ]$
$backupWFM \triangleq [\ src \mapsto backupPlace,\ dst \mapsto here,\ fid \mapsto root,\ aid \mapsto actId,\ source \mapsto source,\ target \mapsto here,\ isAdopter \mapsto isAdopter,\ adoptedFID \mapsto adoptedFID,\ type \mapsto \text{"backupLiveDone"}\ ]$
IN $\wedge\ SetActionNameAndDepth(\langle \text{"MasterLiveDone"}, here\rangle)$
$\wedge\ activity \neq C!NotActivity$
$\wedge\ fstates[activity.fid].here = here$
Technically, we should check the condition $rootPlace \notin killed$

```
                    if success is true. we should communicate with the backup normally.
                    the backup then should reject the request and notify the requester
                    that the master has changed, so that we redirect the call to the
                    new master.
                  ∧ IF success ∧ submit ∧ rootPlace ∉ killed
                      THEN ∧ C!ReplaceMsg(msg, [ mid ↦ seq.mseq,
                                                 src ↦ here,
                                                 dst ↦ backupPlace,
                                                 source ↦ source,
                                                 target ↦ here,
                                                 fid ↦ root,
                                                 aid ↦ actId,
                                                 type ↦ "backupLive",
                                                 isAdopter ↦ isAdopter,
                                                 adoptedFID ↦ adoptedFID])
                           ∧ waitForMsgs' = (waitForMsgs \ {masterWFM}) ∪ {backupWFM}
                           ∧ C!IncrMSEQ(1)
                           ∧ pendingAct' = pendingAct
                      ELSE IF success ∧ rootPlace ∉ killed
                      THEN ∧ C!RecvMsg(msg)
                           ∧ pendingAct' = pendingAct \ {activity}
                           ∧ seq' = seq
                           ∧ waitForMsgs' = waitForMsgs \ {masterWFM}
                      ELSE ∧ C!ReplaceMsg(msg, [ mid ↦ seq.mseq,
                                                 src ↦ here,
                                                 dst ↦ C!GetBackup(rootPlace),
                                                 source ↦ source,
                                                 target ↦ here,
                                                 fid ↦ root,
                                                 type ↦ "backupGetAdopter",
                                                 aid ↦ actId,
                                                 finishEnd ↦ FALSE,
                                                 actionType ↦ "live"])
                           ∧ waitForMsgs' = waitForMsgs \ {masterWFM}
                                we don't expect backup to die
                                so we don't add
                                backupGetAdopterDone in waitForMsgs
                           ∧ C!IncrMSEQ(1)
                           ∧ pendingAct' = pendingAct
  ∧ UNCHANGED ⟨convertSet, adoptSet, mastersStatus, fstates, pstate,
               thrds, killed, fmasters, fbackups, blockedThrds, runningThrds⟩

MasterCompletedDone ≜
  ∧ pstate = "running"
  ∧ msgs ≠ {}
```

$$
\begin{array}{l}
\land \text{LET } msg \triangleq C!FindMSG(\text{“masterCompletedDone”}) \\
\quad \text{IN} \quad \land msg \neq C!NotMessage \\
\qquad \land \text{LET } here \triangleq msg.dst \\
\qquad\qquad root \triangleq msg.fid \\
\qquad\qquad success \triangleq msg.success \\
\qquad\qquad rootPlace \triangleq C!GetFinishHome(root) \\
\qquad\qquad isAdopter \triangleq msg.isAdopter \\
\qquad\qquad backupPlace \triangleq msg.backupPlace \\
\qquad\qquad finishEnd \triangleq msg.finishEnd \\
\qquad\qquad masterWFM \triangleq [\ src \mapsto rootPlace, \\
\qquad\qquad\qquad\qquad dst \mapsto here, \\
\qquad\qquad\qquad\qquad target \mapsto here, \\
\qquad\qquad\qquad\qquad fid \mapsto root, \\
\qquad\qquad\qquad\qquad isAdopter \mapsto isAdopter, \\
\qquad\qquad\qquad\qquad type \mapsto \text{“masterCompletedDone”}\ ] \\
\qquad\qquad backupWFM \triangleq [\ src \mapsto backupPlace, \\
\qquad\qquad\qquad\qquad dst \mapsto here, \\
\qquad\qquad\qquad\qquad fid \mapsto root, \\
\qquad\qquad\qquad\qquad target \mapsto here, \\
\qquad\qquad\qquad\qquad isAdopter \mapsto isAdopter, \\
\qquad\qquad\qquad\qquad type \mapsto \text{“backupCompletedDone”}\ ] \\
\qquad \text{IN} \quad \land SetActionNameAndDepth(\langle \text{“MasterCompletedDone”}, here \rangle) \\
\qquad\qquad \text{Technically, we should check the condition } rootPlace \notin killed \\
\qquad\qquad \text{if success is true. we should communicate with the backup normally.} \\
\qquad\qquad \text{the backup then should reject the request and notify the requester} \\
\qquad\qquad \text{that the master has changed, so that we redirect the call to the} \\
\qquad\qquad \text{new master.} \\
\qquad\qquad \land \text{IF } success \land rootPlace \notin killed \\
\qquad\qquad\quad \text{THEN} \quad \land C!ReplaceMsg(msg, [mid \mapsto seq.mseq, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad src \mapsto here, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad dst \mapsto backupPlace, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad target \mapsto here, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad fid \mapsto root, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad type \mapsto \text{“backupCompleted”}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad finishEnd \mapsto finishEnd, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad isAdopter \mapsto isAdopter]) \\
\qquad\qquad\qquad\qquad \land \text{IF } finishEnd \text{ THEN } waitForMsgs' = (waitForMsgs \setminus \{masterWFM\}) \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{ELSE } waitForMsgs' = (waitForMsgs \setminus \{masterWFM\}) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \cup \{backupWFM\} \\
\qquad\qquad\qquad\qquad \land C!IncrMSEQ(1) \\
\qquad\qquad\quad \text{ELSE} \quad \land C!ReplaceMsg(msg, [\ mid \mapsto seq.mseq, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad src \mapsto here, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad dst \mapsto C!GetBackup(rootPlace), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad source \mapsto C!NotPlace, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad target \mapsto here,
\end{array}
$$

```
                                                    fid       ↦ root,
                                                    type      ↦ "backupGetAdopter",
                                                    aid       ↦ C!NotActivity.aid,
                                                    finishEnd ↦ FALSE,
                                                    actionType ↦ "completed"])
                          ∧ waitForMsgs' = waitForMsgs \ {masterWFM}
                                we don't expect backup to die
                                so we don't add backupGetAdopterDone
                                in waitForMsgs
                          ∧ C!IncrMSEQ(1)
    ∧ UNCHANGED ⟨convertSet, adoptSet, mastersStatus, fstates, pstate,
                 thrds, pendingAct, killed, fmasters, fbackups,
                 blockedThrds, runningThrds⟩

──────────────────────────────────────────────────────────────────────────

GetAdopterDone ≜
  ∧ pstate = "running"
  ∧ msgs ≠ {}
  ∧ LET msg ≜ C!FindMSG("backupGetAdopterDone")
    IN  ∧ msg ≠ C!NotMessage
        ∧ LET here ≜ msg.dst
                actionType ≜ msg.actionType
                adoptedRoot ≜ msg.adoptedRoot
                adoptedRootPlace ≜ C!GetFinishHome(msg.adoptedRoot)
                adoptedFID ≜ msg.fid
          IN  ∧ SetActionNameAndDepth(⟨"GetAdopterDone", here⟩)
              ∧ IF actionType = "transit"
                THEN ∧ C!ReplaceMsg(msg, [ mid    ↦ seq.mseq,
                                           src    ↦ here,
                                           dst    ↦ adoptedRootPlace,
                                           target ↦ msg.target,
                                           fid    ↦ adoptedRoot,
                                           type   ↦ "adopterTransit",
                                           adoptedFID ↦ adoptedFID])
                     ∧ C!IncrMSEQ(1)
                ELSE IF actionType = "live"
                THEN ∧ C!ReplaceMsg(msg, [mid ↦ seq.mseq,
                                           src    ↦ here,
                                           dst    ↦ adoptedRootPlace,
                                           source ↦ msg.source,
                                           target ↦ msg.target,
                                           fid    ↦ adoptedRoot,
                                           aid    ↦ msg.aid,
                                           type   ↦ "adopterLive",
                                           adoptedFID ↦ adoptedFID])
```

$\land\ C!IncrMSEQ(1)$
ELSE IF $actionType =$ "completed"
THEN $\land\ C!ReplaceMsg(msg, [mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto adoptedRootPlace,$
$target \mapsto msg.target,$
$fid \mapsto adoptedRoot,$
$finishEnd \mapsto msg.finishEnd,$
$type \mapsto$ "adopterCompleted",
$adoptedFID \mapsto adoptedFID])$
$\land\ C!IncrMSEQ(1)$
ELSE FALSE
$\land$ UNCHANGED $\langle fstates, pstate, thrds, killed, pendingAct, fmasters, fbackups, waitForMsgs, mastersStatus, adoptSet, convertSet, blockedThrds, runningThrds\rangle$

$FindBlockedThreadAsyncTerm \triangleq$
LET $tset \triangleq \{t \in blockedThrds :$
$\land\ t.here \notin killed$
$\land\ thrds[t.here][t.tid].status =$ "blocked"
$\land\ thrds[t.here][t.tid].blockingType =$ "AsyncTerm"
$\land$ LET $msg \triangleq C!FindIncomingMSG(t.here,$ "backupCompletedDone")
$top \triangleq Head(thrds[t.here][t.tid].stack)$
$blk \triangleq top.b$
IN $\land\ msg \neq C!NotMessage$
$\land\ PROG[blk].type =$ "async"
$\land\ PROG[blk].mxstmt = top.i$
$\land\ msg.fid = fstates[top.fid].root\}$
IN IF $tset = \{\}$ THEN $C!NotPlaceThread$
ELSE CHOOSE $x \in tset$: TRUE

Terminated finish unblocks its thread
$UnblockTerminateAsync \triangleq$
$\land\ pstate =$ "running"
$\land$ LET $pthrd \triangleq FindBlockedThreadAsyncTerm$
IN $\land\ pthrd \neq C!NotPlaceThread$
$\land$ LET $here \triangleq pthrd.here$
$tid \triangleq pthrd.tid$
$msg \triangleq C!FindIncomingMSG(here,$ "backupCompletedDone")
$success \triangleq msg.success$
$top \triangleq Head(thrds[here][tid].stack)$
$blk \triangleq top.b$
$fid \triangleq top.fid$
$root \triangleq msg.fid$
$rootPlace \triangleq C!GetFinishHome(root)$

```
        IN  ∧ SetActionNameAndDepth(⟨"UnblockTerminateAsync", here,
                                    "success", success⟩)
            ∧ waitForMsgs' = waitForMsgs \ {[src    ↦ rootPlace,
                                            dst    ↦ here,
                                            target ↦ here,
                                            fid    ↦ root,
                                            type   ↦ "backupCompletedDone" ]}
            ∧ IF    Len(thrds[here][tid].stack) = 1
              THEN  ∧ thrds' = [thrds EXCEPT ![here][tid].stack = ⟨⟩,
                                             ![here][tid].status = "idle"]
                    ∧ blockedThrds'  = blockedThrds \ {[here ↦ here, tid ↦ tid]}
                    ∧ runningThrds' = runningThrds
              ELSE  ∧ thrds' = [thrds EXCEPT ![here][tid].stack = Tail(@),
                                             ![here][tid].status = "running"]
                    ∧ blockedThrds'  = blockedThrds \ {[here ↦ here, tid ↦ tid]}
                    ∧ runningThrds' = runningThrds ∪ {[here ↦ here, tid ↦ tid]}
            ∧ IF  blk = 0
              THEN pstate' = "terminated"
              ELSE pstate' = pstate
            ∧ C!RecvMsg(msg)
  ∧ UNCHANGED ⟨convertSet, adoptSet, mastersStatus, fstates, seq,
               killed, pendingAct, fmasters, fbackups⟩

------------------------------------------------------------------------

FindBlockedThreadAuthorizeTransitAsync ≜
  LET tset ≜ {t ∈ blockedThrds :
                ∧ t.here ∉ killed
                ∧ thrds[t.here][t.tid].status = "blocked"
                ∧ thrds[t.here][t.tid].blockingType = "AsyncTransit"
                ∧ C!FindIncomingMSG(t.here, "backupTransitDone") ≠ C!NotMessage }
  IN  IF tset = {} THEN C!NotPlaceThread
      ELSE CHOOSE x ∈ tset : TRUE

AuthorizeTransitAsync ≜
  ∧ pstate = "running"
  ∧ msgs ≠ {}
  ∧ LET pthrd ≜ FindBlockedThreadAuthorizeTransitAsync
    IN  ∧ pthrd ≠ C!NotPlaceThread
        ∧ LET here ≜ pthrd.here
              tid  ≜ pthrd.tid
              msg  ≜ C!FindIncomingMSG(here, "backupTransitDone")
              success ≜ msg.success
              top  ≜ Head(thrds[here][tid].stack)
              tail ≜ Tail(thrds[here][tid].stack)
              lstStmt  ≜ top.i
```

```
                curStmt ≜ top.i + 1
                blk ≜ top.b
                root ≜ msg.fid
                fid ≜ top.fid
                rootPlace ≜ C!GetFinishHome(root)
                backupPlace ≜ msg.src
                nested ≜ PROG[blk].stmts[curStmt]
                asyncDst ≜ PROG[nested].dst
                realFID ≜ IF msg.adoptedFID ≠ C!NotID THEN msg.adoptedFID ELSE root
            IN  ∧ SetActionNameAndDepth(⟨"AuthorizeTransitAsync", here, "to",
                                         asyncDst, "success", success⟩)
                ∧ C!ReplaceMsg(msg,
                               [mid  ↦ seq.mseq,
                                src  ↦ here,
                                dst  ↦ asyncDst,
                                type ↦ "async",
                                fid  ↦ realFID,
                                b    ↦ nested])
                ∧ C!IncrMSEQ(1)
                ∧ thrds' = [thrds EXCEPT  ![here][tid].status = "running",
                                          ![here][tid].stack =
                                                    ⟨[ b   ↦ top.b,
                                                       i   ↦ curStmt,
                                                       fid ↦ fid]
                                                    ⟩ ∘ tail]
                ∧ blockedThrds'  = blockedThrds \ {[here ↦ here, tid ↦ tid]}
                ∧ runningThrds' = runningThrds ∪ {[here ↦ here, tid ↦ tid]}
                ∧ waitForMsgs' = waitForMsgs \ {[type     ↦ "backupTransitDone",
                                                 dst      ↦ msg.dst,
                                                 fid      ↦ msg.fid,
                                                 src      ↦ backupPlace,
                                                 target   ↦ asyncDst,
                                                 isAdopter ↦ msg.isAdopter,
                                                 adoptedFID ↦ msg.adoptedFID]}
    ∧ UNCHANGED ⟨convertSet, adoptSet, mastersStatus, fstates, pstate,
                 killed, pendingAct, fmasters, fbackups⟩

AuthorizeReceivedAsync ≜
    ∧ pstate = "running"
    ∧ pendingAct ≠ {}
    ∧ msgs ≠ {}
    ∧ LET msg ≜ C!FindMSG("backupLiveDone")
      IN  ∧ msg ≠ C!NotMessage
          ∧ LET backupPlace ≜ msg.src
                here ≜ msg.dst
```

```
                actId ≜ msg.aid
                activity ≜ FindPendingActivity(actId)
                root ≜ msg.fid
                success ≜ msg.success
                rootPlace ≜ C!GetFinishHome(root)
          IN    ∧ SetActionNameAndDepth(⟨"AuthorizeReceivedAsync", here, "success", success⟩)
                ∧ activity ≠ C!NotActivity
                ∧ fstates[activity.fid].here = here
                ∧ waitForMsgs' = waitForMsgs \ {[src    ↦ backupPlace,
                                                 dst    ↦ here,
                                                 fid    ↦ root,
                                                 aid    ↦ actId,
                                                 source ↦ msg.source,
                                                 target ↦ msg.target,
                                                 type   ↦ "backupLiveDone",
                                                 isAdopter  ↦ msg.isAdopter,
                                                 adoptedFID ↦ msg.adoptedFID ]}
                ∧ C!RecvMsg(msg)
                ∧ pendingAct' = pendingAct \ {activity}
                ∧ LET idleThrd ≜ FindIdleThread(here)
                      stkEntry ≜ [b ↦ activity.b, i ↦ C!I_START, fid ↦ activity.fid]
                  IN  ∧ thrds' = [thrds EXCEPT ![here][idleThrd.tid].stack = ⟨stkEntry⟩,
                                               ![here][idleThrd.tid].status = "running"]
                      ∧ runningThrds' = runningThrds ∪ {[here ↦ here, tid ↦ idleThrd.tid]}
      ∧ UNCHANGED ⟨convertSet, adoptSet, mastersStatus, fstates, pstate, seq,
                   killed, fmasters, fbackups, blockedThrds⟩

──────────────────────────────────────────────────────────────────────────

FindBlockedThreadStopFinish(here, root) ≜
  LET tset ≜ {t ∈ blockedThrds :
                ∧ here = t.here
                ∧ t.here ∉ killed
                ∧ thrds[t.here][t.tid].status = "blocked"
                ∧ thrds[t.here][t.tid].blockingType = "FinishEnd"
                ∧ LET top ≜ Head(thrds[t.here][t.tid].stack)
                      fid ≜ top.fid
                      blk ≜ top.b
                  IN  ∧ PROG[blk].type = "finish"
                      ∧ PROG[blk].mxstmt = top.i
                      ∧ root = fid }
  IN  IF tset = {} THEN C!NotPlaceThread
      ELSE CHOOSE x ∈ tset : TRUE

Terminated finish unblocks its thread
UnblockStopFinish(here, tid, fid, blk) ≜
```

```
  ∧ IF     Len(thrds[here][tid].stack) = 1
    THEN   ∧ thrds' = [thrds EXCEPT ![here][tid].stack = ⟨⟩,
                                    ![here][tid].status = "idle"]
           ∧ blockedThrds'  = blockedThrds \ {[here ↦ here, tid ↦ tid]}
           ∧ runningThrds' = runningThrds
    ELSE   ∧ thrds' = [thrds EXCEPT ![here][tid].stack = Tail(@),
                                    ![here][tid].status = "running"]
           ∧ blockedThrds'  = blockedThrds \ {[here ↦ here, tid ↦ tid]}
           ∧ runningThrds' = runningThrds ∪ {[here ↦ here, tid ↦ tid]}
  ∧ IF   blk = 0
    THEN pstate' = "terminated"
    ELSE pstate' = pstate

ReleaseRootFinish ≜
  ∧ pstate = "running"
  ∧ msgs ≠ {}
  ∧ blockedThrds ≠ {}
  ∧ LET msg ≜ C!FindMSG("releaseFinish")
    IN   ∧ msg ≠ C!NotMessage
         ∧ LET here ≜ msg.dst
                root ≜ msg.fid
                pthrd ≜ FindBlockedThreadStopFinish(here, root)
                tid ≜ pthrd.tid
                top ≜ Head(thrds[here][tid].stack)
                blk ≜ top.b
           IN   ∧ msg ≠ C!NotMessage
                ∧ SetActionNameAndDepth(⟨"ReleaseRootFinish", here⟩)
                ∧ C!RecvMsg(msg)
                ∧ fstates' = [fstates EXCEPT ![root].status = "forgotten"]
                ∧ waitForMsgs' = waitForMsgs \ {[src  ↦ here,
                                                 dst  ↦ here,
                                                 fid  ↦ root,
                                                 type ↦ "releaseFinish" ]}
                ∧ UnblockStopFinish(here, tid, root, blk)
  ∧ UNCHANGED ⟨convertSet, adoptSet, mastersStatus, seq,
               killed, pendingAct,  fmasters, fbackups⟩
```

Finish master replica actions

```
MasterTransit ≜
  ∧ pstate = "running"
  ∧ msgs ≠ {}
  ∧ LET msg ≜ C!FindMSG("masterTransit")
    IN   ∧ msg ≠ C!NotMessage
         ∧ LET here ≜ msg.dst
```

```
                fid ≜ msg.fid
                src ≜ msg.src
                target ≜ msg.target
                backupPlace ≜ C!GetBackup(here)
          IN    ∧ SetActionNameAndDepth(⟨"MasterTransit", here⟩)
                ∧ mastersStatus[here].status = "running"
                ∧ src ≠ C!NotPlace
                ∧ fid ≠ C!NotID
                ∧ fstates[fid].here      = here
                ∧ LET submit ≜ src ∉ killed ∧ target ∉ killed
                  IN    ∧ IF submit
                           THEN IF fmasters[fid].id = C!NotID
                                   THEN fmasters' = [fmasters EXCEPT ![fid].id = fid,
                                                                     ![fid].backupPlace = backupPlace,
                                                                     ![fid].transit[src][target] = @ + 1,
                                                                     ![fid].numActive  = @ + 2,
                                                                     ![fid].live[here]     = 1]
                                   ELSE fmasters' = [fmasters EXCEPT ![fid].transit[src][target] = @ + 1,
                                                                     ![fid].numActive = @ + 1]
                                ELSE fmasters' = fmasters
                        ∧ IF src ∈ killed
                           THEN ∧ C!RecvMsg(msg)
                                ∧ seq' = seq
                           ELSE ∧ C!ReplaceMsg(msg, [ mid       ↦ seq.mseq,
                                                      src       ↦ here,
                                                      dst       ↦ src,
                                                      target    ↦ target,
                                                      fid       ↦ fid,
                                                      type      ↦ "masterTransitDone",
                                                      submit    ↦ submit,
                                                      success   ↦ TRUE,
                                                      isAdopter ↦ FALSE,
                                                      adoptedFID ↦ C!NotID,
                                                      backupPlace ↦ backupPlace])
                                ∧ C!IncrMSEQ(1)
          ∧ UNCHANGED ⟨waitForMsgs, convertSet, adoptSet, mastersStatus,
                       fstates, pstate, thrds, killed, pendingAct, fbackups,
                       blockedThrds, runningThrds⟩

------------------------------------------------------------------------

MasterLive ≜
  ∧ pstate = "running"
  ∧ msgs ≠ {}
  ∧ LET msg ≜ C!FindMSG("masterLive")
    IN    ∧ msg ≠ C!NotMessage
```

```
    ∧ LET here ≜ msg.dst
          fid ≜ msg.fid
          source ≜ msg.source
          target ≜ msg.target   msg.target = msg.src
          backupPlace ≜ C!GetBackup(here)
      IN  ∧ SetActionNameAndDepth(⟨"MasterLive", here⟩)
          ∧ fid ≠ C!NotID
          ∧ fstates[fid].here = here
          ∧ mastersStatus[here].status = "running"
          ∧ target = msg.src
          ∧ LET submit ≜ source ∉ killed ∧ target ∉ killed
            IN  ∧ IF submit
                   THEN ∧ fmasters[fid].transit[source][target] > 0
                        ∧ fmasters' = [fmasters EXCEPT ![fid].transit[source][target] = @ − 1,
                                                       ![fid].live[target] = @ + 1]
                   ELSE ∧ fmasters' = fmasters
                ∧ IF target ∈ killed
                   THEN ∧ C!RecvMsg(msg)
                        ∧ seq' = seq
                   ELSE ∧ C!ReplaceMsg(msg, [ mid ↦ seq.mseq,
                                              src ↦ here,
                                              dst ↦ target,
                                              source ↦ source,
                                              target ↦ target,
                                              fid ↦ fid,
                                              aid ↦ msg.aid,
                                              type ↦ "masterLiveDone",
                                              submit ↦ submit,
                                              success ↦ TRUE,
                                              isAdopter ↦ FALSE,
                                              adoptedFID ↦ C!NotID,
                                              backupPlace ↦ backupPlace])
                ∧ C!IncrMSEQ(1)
    ∧ UNCHANGED ⟨convertSet, adoptSet, mastersStatus, fstates, pstate,
                 thrds, waitForMsgs, killed, pendingAct, fbackups,
                 blockedThrds, runningThrds⟩

──────────────────────────────────────────────────────────────────────

MasterCompleted ≜
  ∧ pstate = "running"
  ∧ msgs ≠ {}
  ∧ LET msg ≜ C!FindMSG("masterCompleted")
    IN  ∧ msg ≠ C!NotMessage
        ∧ LET here ≜ msg.dst
              mid ≜ msg.mid
```

```
    fid    ≜ msg.fid
    src    ≜ msg.src
    target ≜ msg.target
    backupPlace ≜ C!GetBackup(here)
    finishEnd ≜ msg.finishEnd
IN  ∧ SetActionNameAndDepth(⟨"MasterCompleted", here⟩)
    ∧ backupPlace ≠ C!NotPlace
    ∧ fid ≠ C!NotID
    ∧ fstates[fid].here = here
    ∧ target = src
    ∧ mastersStatus[here].status = "running"
    ∧ IF (fmasters[fid].live[target] > 0 ∧ fmasters[fid].numActive > 0)
      THEN ∧ fmasters' = [fmasters EXCEPT ![fid].live[target] = @ − 1,
                                          ![fid].numActive    = @ − 1,
                                          ![fid].isReleased   =
                                              (fmasters[fid].numActive = 1)]
      ELSE ∧ target ∈ killed
           ∧ fmasters' = fmasters
    ∧ IF (fmasters'[fid].numActive = 0 ∧ src ∉ killed)
      THEN ∧ C!ReplaceMsgSet(msg, {[mid  ↦ seq.mseq,
                                    src  ↦ here,
                                    dst  ↦ src,
                                    target ↦ target,
                                    fid  ↦ fid,
                                    type ↦ "masterCompletedDone",
                                    success ↦ TRUE,
                                    isAdopter ↦ FALSE,
                                    finishEnd ↦ finishEnd,
                                    backupPlace ↦ backupPlace],
                                   [mid  ↦ seq.mseq + 1,
                                    src  ↦ here,
                                    dst  ↦ here,
                                    fid  ↦ fid,
                                    type ↦ "releaseFinish"]})
           ∧ C!IncrMSEQ(2)
      ELSE IF fmasters'[fid].numActive = 0
      THEN ∧ C!ReplaceMsg(msg, [mid ↦ seq.mseq,
                                src ↦ here,
                                dst ↦ here,
                                fid ↦ fid,
                                type ↦ "releaseFinish"])
           ∧ C!IncrMSEQ(1)
      ELSE IF src ∉ killed
      THEN ∧ C!ReplaceMsg(msg, [mid  ↦ seq.mseq,
                                src  ↦ here,
```

```
                                                    dst       ↦ src,
                                                    target    ↦ target,
                                                    fid       ↦ fid,
                                                    type      ↦ "masterCompletedDone",
                                                    success   ↦ TRUE,
                                                    isAdopter ↦ FALSE,
                                                    finishEnd ↦ finishEnd,
                                                    backupPlace ↦ backupPlace])
                                  ∧ C!IncrMSEQ(1)
                          ELSE    ∧ C!RecvMsg(msg)
                                  ∧ seq' = seq
    ∧ UNCHANGED ⟨convertSet, adoptSet, mastersStatus, fstates, pstate,
                 thrds, killed, pendingAct, fbackups, waitForMsgs,
                 blockedThrds, runningThrds⟩
```

Adopting *Finish* master replica actions

```
AdopterTransit ≜
  ∧ pstate = "running"
  ∧ msgs ≠ {}
  ∧ LET msg ≜ C!FindMSG("adopterTransit")
    IN  ∧ msg ≠ C!NotMessage
        ∧ LET here ≜ msg.dst
              fid ≜ msg.fid
              src ≜ msg.src
              target ≜ msg.target
              backupPlace ≜ C!GetBackup(here)
              adoptedFID ≜ msg.adoptedFID
          IN  ∧ SetActionNameAndDepth(⟨"AdopterTransit", here⟩)
              ∧ mastersStatus[here].status = "running"
              ∧ fid ≠ C!NotID
              ∧ fstates[fid].here      = here
              ∧ LET submit ≜ src ∉ killed ∧ target ∉ killed
                IN  ∧ IF submit
                      THEN ∧ fmasters' = [fmasters EXCEPT ![fid].transitAdopted[src][target] = @ + 1,
                                                          ![fid].numActive = @ + 1]
                      ELSE ∧ fmasters' = fmasters
                    ∧ C!ReplaceMsg(msg, [ mid    ↦ seq.mseq,
                                          src    ↦ here,
                                          dst    ↦ src,
                                          target ↦ target,
                                          fid    ↦ fid,
                                          type   ↦ "masterTransitDone",
                                          submit ↦ submit,
                                          success ↦ TRUE,
```

```
                                                backupPlace  ↦ backupPlace,
                                                  isAdopter  ↦ TRUE,
                                                 adoptedFID  ↦ adoptedFID ])
                                  ∧ C!IncrMSEQ(1)
        ∧ UNCHANGED ⟨convertSet, adoptSet, mastersStatus, fstates, pstate,
                     thrds, killed, pendingAct, fbackups, waitForMsgs,
                     blockedThrds, runningThrds⟩

AdopterLive ≜
  ∧ pstate = "running"
  ∧ msgs ≠ {}
  ∧ LET msg ≜ C!FindMSG("adopterLive")
    IN  ∧ msg ≠ C!NotMessage
        ∧ LET here ≜ msg.dst
              fid ≜ msg.fid
              source ≜ msg.source
              target ≜ msg.target
              backupPlace ≜ C!GetBackup(here)
              adoptedFID ≜ msg.adoptedFID
          IN  ∧ SetActionNameAndDepth(⟨"AdopterLive", here⟩)
              ∧ fid ≠ C!NotID
              ∧ backupPlace ≠ C!NotPlace
              ∧ fstates[fid].here = here
              ∧ mastersStatus[here].status = "running"
              ∧ target = msg.src
              ∧ LET submit ≜ source ∉ killed ∧ target ∉ killed
                IN  ∧ IF submit
                      THEN ∧ fmasters[fid].transitAdopted[source][target] > 0
                           ∧ fmasters′ = [fmasters EXCEPT ![fid].transitAdopted[source][target] = @ −
                                                          ![fid].liveAdopted[target] = @ + 1]
                      ELSE fmasters′ = fmasters
                    ∧ C!ReplaceMsg(msg, [ mid     ↦ seq.mseq,
                                          src     ↦ here,
                                          dst     ↦ target,
                                          source  ↦ source,
                                          target  ↦ target,
                                          fid     ↦ fid,
                                          aid     ↦ msg.aid,
                                          type    ↦ "masterLiveDone",
                                          submit  ↦ submit,
                                          success ↦ TRUE,
                                          isAdopter ↦ TRUE,
                                          adoptedFID ↦ adoptedFID,
                                          backupPlace ↦ backupPlace])
                    ∧ C!IncrMSEQ(1)
```

```
  ∧ UNCHANGED ⟨convertSet, adoptSet, mastersStatus, fstates, pstate, waitForMsgs,
               thrds, waitForMsgs, killed, pendingAct, fbackups,
               blockedThrds, runningThrds⟩

AdopterCompleted ≜
  ∧ pstate = "running"
  ∧ msgs ≠ {}
  ∧ LET msg ≜ C!FindMSG("adopterCompleted")
    IN  ∧ msg ≠ C!NotMessage
        ∧ LET here ≜ msg.dst
              mid ≜ msg.mid
              fid ≜ msg.fid
              src ≜ msg.src
              target ≜ msg.target
              backupPlace ≜ C!GetBackup(here)
              finishEnd ≜ msg.finishEnd
          IN  ∧ SetActionNameAndDepth(⟨"AdopterCompleted", here⟩)
              ∧ mastersStatus[here].status = "running"
              ∧ backupPlace ≠ C!NotPlace
              ∧ fid ≠ C!NotID
              ∧ fstates[fid].here = here
              ∧ fmasters[fid].liveAdopted[target] > 0
              ∧ fmasters[fid].numActive > 0
              ∧ fmasters' = [fmasters EXCEPT ![fid].liveAdopted[target] = @ − 1,
                                             ![fid].numActive           = @ − 1,
                                             ![fid].isReleased          = (fmasters[fid].numActive = 1)]
              ∧ IF fmasters'[fid].numActive = 0
                THEN ∧ C!ReplaceMsgSet(msg, {[mid        ↦ seq.mseq,
                                              src        ↦ here,
                                              dst        ↦ src,
                                              target     ↦ target,
                                              fid        ↦ fid,
                                              type       ↦ "masterCompletedDone",
                                              success    ↦ TRUE,
                                              isAdopter  ↦ TRUE,
                                              finishEnd  ↦ finishEnd,
                                              backupPlace ↦ backupPlace],
                                             [mid        ↦ seq.mseq + 1,
                                              src        ↦ here,
                                              dst        ↦ here,
                                              fid        ↦ fid,
                                              type       ↦ "releaseFinish"]})
                     ∧ C!IncrMSEQ(2)
                ELSE IF finishEnd
                     THEN ∧ C!RecvMsg(msg)
```

$\land\ seq' = seq$

ELSE $\land\ C!ReplaceMsg(msg, [mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto src,$
$target \mapsto target,$
$fid \mapsto fid,$
$type \mapsto$ "masterCompletedDone",
$success \mapsto$ TRUE,
$isAdopter \mapsto$ TRUE,
$finishEnd \mapsto finishEnd,$
$backupPlace \mapsto backupPlace])$

$\land\ C!IncrMSEQ(1)$

$\land$ UNCHANGED $\langle convertSet, adoptSet, mastersStatus, fstates, pstate,$
$thrds, waitForMsgs, killed, pendingAct, fbackups, waitForMsgs,$
$blockedThrds, runningThrds\rangle$

Finish backup replica actions

$BackupGetAdopter \triangleq$
$\land\ pstate =$ "running"
$\land\ msgs \neq \{\}$
$\land$ LET $msg \triangleq C!FindMSG($"backupGetAdopter"$)$
IN $\land\ msg \neq C!NotMessage$
$\land$ LET $here \triangleq msg.dst$
$fid \triangleq msg.fid$
$src \triangleq msg.src$
$actionType \triangleq msg.actionType$
$source \triangleq msg.source$
$target \triangleq msg.target$
IN $\land\ SetActionNameAndDepth(\langle$"BackupGetAdopter", $here\rangle)$
$\land\ fbackups[fid].isAdopted =$ TRUE
$\land$ IF $src \in killed \lor msg.dst \in killed$
THEN $\land\ C!RecvMsg(msg)$
$\land\ seq' = seq$
ELSE $\land\ C!ReplaceMsg(msg, [\ mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto src,$
$source \mapsto source,$
$target \mapsto target,$
$fid \mapsto fid,$
$adoptedRoot \mapsto fbackups[fid].adoptedRoot,$
$actionType \mapsto actionType,$
$aid \mapsto msg.aid,$
$finishEnd \mapsto msg.finishEnd,$
$type \mapsto$ "backupGetAdopterDone"$])$

```
                                    ∧ C!IncrMSEQ(1)
  ∧ UNCHANGED ⟨fstates, pstate, thrds, killed, pendingAct, fmasters,
                fbackups, waitForMsgs, mastersStatus, adoptSet, convertSet,
                blockedThrds, runningThrds⟩

BackupTransit ≜
  ∧ pstate = "running"
  ∧ msgs ≠ {}
  ∧ LET msg ≜ C!FindMSG("backupTransit")
    IN  ∧ msg ≠ C!NotMessage
        ∧ LET here ≜ msg.dst
              fid ≜ msg.fid
              src ≜ msg.src
              target ≜ msg.target
              isAdopter ≜ msg.isAdopter
              adoptedFID ≜ msg.adoptedFID
          IN  ∧ SetActionNameAndDepth(⟨"BackupTransit", here⟩)
              ∧ fmasters[fid].backupPlace = here
              ∧ IF ¬ isAdopter ∧ ¬fbackups[fid].isAdopted
                THEN IF fbackups[fid].id = C!NotID
                     THEN fbackups' = [fbackups EXCEPT ![fid].id = fid,
                                                       ![fid].transit[src][target] = @ + 1,
                                                       ![fid].live[src] = 1,
                                                       ![fid].numActive = @ + 2]
                     ELSE fbackups' = [fbackups EXCEPT ![fid].transit[src][target] = @ + 1,
                                                       ![fid].numActive = @ + 1]
                ELSE fbackups' = fbackups
              We don't have transitAdopted at the backups!!! fbackups' = [ fbackups
              EXCEPT ![fid].transitAdopted[src][target] = @ + 1,
                     ![fid].numActive = @ + 1 ]
              ∧ IF fbackups[fid].isAdopted  Change to the path of adopterTransit
                THEN ∧ C!ReplaceMsg(msg, [ mid ↦ seq.mseq,
                                           src ↦ here,
                                           dst ↦ src,
                                           target ↦ target,
                                           fid ↦ fid,
                                           type ↦ "masterTransitDone",
                                           isAdopter ↦ FALSE,
                                           adoptedFID ↦ C!NotID,
                                           backupPlace ↦ C!NotPlace,
                                           submit ↦ FALSE,
                                           success ↦ FALSE] )
                     ∧ C!IncrMSEQ(1)
                ELSE IF src ∈ killed
                THEN ∧ C!RecvMsg(msg)
```

$\land\ seq' = seq$

ELSE $\land\ C!ReplaceMsg(msg, [\ mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto src,$
$target \mapsto target,$
$fid \mapsto fid,$
$type \mapsto$ "backupTransitDone",
$success \mapsto$ TRUE,
$isAdopter \mapsto isAdopter,$
$adoptedFID \mapsto adoptedFID])$

$\land\ C!IncrMSEQ(1)$

$\land$ UNCHANGED $\langle convertSet, adoptSet, mastersStatus, fstates, pstate,$
$thrds, killed, pendingAct, fmasters, waitForMsgs,$
$blockedThrds, runningThrds\rangle$

$BackupLive \triangleq$
$\land\ pstate =$ "running"
$\land\ msgs \neq \{\}$
$\land$ LET $msg \triangleq C!FindMSG($"backupLive"$)$
IN $\land\ msg \neq C!NotMessage$
$\land$ LET $here \triangleq msg.dst$
$fid \triangleq msg.fid$
$src \triangleq msg.src$
$source \triangleq msg.source$
$target \triangleq msg.target$
$isAdopter \triangleq msg.isAdopter$
$adoptedFID \triangleq msg.adoptedFID$
IN $\land\ SetActionNameAndDepth(\langle$"BackupLive", $here\rangle)$
$\land\ fmasters[fid].backupPlace = here$
$\land$ IF $\neg\ isAdopter \land \neg fbackups[fid].isAdopted$
THEN $\land\ fbackups[fid].transit[source][target] > 0$
$\land\ fbackups' = [fbackups$ EXCEPT $![fid].transit[source][target] = @ - 1,$
$![fid].live[target] = @ + 1]$
ELSE $\land\ fbackups' = fbackups$

We don't have *transitAdopted* at the backups!!!!
$\land\ fbackups[fid].transitAdopted[source][target] > 0$
$\land\ fbackups' = [\ fbackups$ EXCEPT $![fid].transitAdopted[source][target] = @ -$
$1, ![fid].liveAdopted[target] = @ + 1\ ]$

$\land$ IF $fbackups[fid].isAdopted$ Change to the path of *adopterLive*
THEN $\land\ C!ReplaceMsg(msg, [\ mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto src,$
$source \mapsto source,$
$target \mapsto target,$
$fid \mapsto fid,$

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad aid \mapsto msg.aid, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad type \mapsto \text{“masterLiveDone”}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad submit \mapsto \text{FALSE}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad success \mapsto \text{FALSE}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad isAdopter \mapsto \text{FALSE}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad adoptedFID \mapsto C!NotID, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad backupPlace \mapsto C!NotPlace]) \\
\qquad\qquad\qquad \land C!IncrMSEQ(1) \\
\qquad\qquad \text{ELSE IF } src \in killed \\
\qquad\qquad \text{THEN } \land C!RecvMsg(msg) \\
\qquad\qquad\qquad \land seq' = seq \\
\qquad\qquad \text{ELSE } \land C!ReplaceMsg(msg, [\ mid \mapsto seq.mseq, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad src \mapsto here, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad dst \mapsto src, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad target \mapsto target, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad source \mapsto source, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad fid \mapsto fid, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad aid \mapsto msg.aid, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad type \mapsto \text{“backupLiveDone”}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad success \mapsto \text{TRUE}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad isAdopter \mapsto isAdopter, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad adoptedFID \mapsto adoptedFID]) \\
\qquad\qquad\qquad \land C!IncrMSEQ(1) \\
\land \text{UNCHANGED } \langle convertSet, fstates, pstate, thrds, pendingAct, fmasters, waitForMsgs, \\
\qquad\qquad blockedThrds, runningThrds, killed, adoptSet, mastersStatus\rangle
\end{array}
$$

$$
\begin{array}{l}
BackupCompleted \triangleq \\
\quad \land pstate = \text{“running”} \\
\quad \land msgs \neq \{\} \\
\quad \land \text{LET } msg \triangleq C!FindMSG(\text{“backupCompleted”}) \\
\qquad \text{IN } \land msg \neq C!NotMessage \\
\qquad\qquad \land \text{LET } here \triangleq msg.dst \\
\qquad\qquad\qquad fid \triangleq msg.fid \\
\qquad\qquad\qquad src \triangleq msg.src \\
\qquad\qquad\qquad target \triangleq msg.target \\
\qquad\qquad\qquad isAdopter \triangleq msg.isAdopter \\
\qquad\qquad\qquad finishEnd \triangleq msg.finishEnd \\
\qquad\qquad \text{IN } \land fmasters[fid].backupPlace = here \\
\qquad\qquad\qquad \land SetActionNameAndDepth(\langle \text{“BackupCompleted”}, here\rangle) \\
\qquad\qquad\qquad \land \text{IF } \neg\ isAdopter \land \neg fbackups[fid].isAdopted \\
\qquad\qquad\qquad\quad \text{THEN } \land fbackups[fid].live[target] > 0 \\
\qquad\qquad\qquad\qquad\quad \land fbackups[fid].numActive > 0 \\
\qquad\qquad\qquad\qquad\quad \land fbackups' = [fbackups \text{ EXCEPT } ![fid].live[target] = @ - 1, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad ![fid].numActive = @ - 1] \\
\qquad\qquad\qquad\quad \text{ELSE } \land fbackups' = fbackups
\end{array}
$$

```
                ∧ IF fbackups[fid].isAdopted    Change to the path of adopterCompleted
                  THEN ∧ C!ReplaceMsg(msg, [ mid       ↦ seq.mseq,
                                             src       ↦ here,
                                             dst       ↦ src,
                                             target    ↦ target,
                                             fid       ↦ fid,
                                             type      ↦ "masterCompletedDone",
                                             success   ↦ FALSE,
                                             isAdopter ↦ FALSE,
                                             finishEnd ↦ FALSE,
                                             backupPlace ↦ C!NotPlace])
                       ∧ C!IncrMSEQ(1)
                  ELSE IF src ∈ killed ∨ finishEnd
                  THEN ∧ C!RecvMsg(msg)
                       ∧ seq' = seq
                  ELSE ∧ C!ReplaceMsg(msg, [ mid       ↦ seq.mseq,
                                             src       ↦ here,
                                             dst       ↦ src,
                                             target    ↦ target,
                                             fid       ↦ fid,
                                             isAdopter ↦ isAdopter,
                                             type      ↦ "backupCompletedDone",
                                             success   ↦ TRUE])
                       ∧ C!IncrMSEQ(1)
            ∧ UNCHANGED ⟨convertSet, adoptSet, mastersStatus, fstates, pstate,
                         thrds, killed, pendingAct, fmasters, waitForMsgs,
                         blockedThrds, runningThrds⟩
```

Finish adoption actions for recovery

```
GetAdoptionSeeker ≜
  IF adoptSet = {} THEN C!NotAdopter
   ELSE CHOOSE m ∈ adoptSet : mastersStatus[m.here].status = "seekAdoption"

SeekAdoption ≜
  ∧ pstate = "running"
  ∧ ∃ p ∈ PLACE : mastersStatus[p].status = "seekAdoption"
  ∧ LET pair ≜ GetAdoptionSeeker
    IN  ∧ pair ≠ C!NotAdopter
        ∧ pair.here ∉ killed
        ∧ LET here ≜ pair.here
              adopter ≜ pair.adopter
              child ≜ pair.child
          IN  ∧ SetActionNameAndDepth(⟨"SeekAdoption", here⟩)
              ∧ fbackups' = [fbackups EXCEPT ![child].isAdopted = TRUE,
```

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad ![child].adoptedRoot = adopter] \\
\qquad\qquad\qquad\qquad \land\ fmasters' = [fmasters\ \text{EXCEPT}\ ![adopter].children = fmasters[adopter].children \setminus \{child\}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad ![adopter].liveAdopted = \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad [p \in PLACE \mapsto fmasters[adopter].liveAdopted[p] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad + fbackups[child].live[p]], \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad ![adopter].transitAdopted = \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad [p \in PLACE \mapsto \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad [q \in PLACE \mapsto fmasters[adopter].transitAdopted[p][q] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad + fbackups[child].transit[p][q]]], \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad ![adopter].numActive = @ + fbackups[child].numActive] \\
\qquad\qquad\qquad\qquad \land\ adoptSet' = adoptSet \setminus \{pair\} \\
\qquad\qquad\qquad\qquad \land\ \text{IF}\ \exists\, m \in adoptSet' : m.here = here \\
\qquad\qquad\qquad\qquad\quad \text{THEN}\ \land\ mastersStatus' = mastersStatus \\
\qquad\qquad\qquad\qquad\quad \text{ELSE}\ \land\ mastersStatus' = [mastersStatus\ \text{EXCEPT}\ ![here].status = \text{“convertDead”}] \\
\qquad\qquad \land\ \text{UNCHANGED}\ \langle fstates, msgs, pstate, seq, thrds, killed, pendingAct, waitForMsgs, \\
\qquad\qquad\qquad\qquad\quad convertSet, blockedThrds, runningThrds\rangle
\end{array}
$$

$$
\begin{array}{l}
GetConvertSeeker \triangleq \\
\quad \text{IF}\ convertSet = \{\}\ \text{THEN}\ C!NotConvTask \\
\quad \text{ELSE}\ \text{CHOOSE}\ m \in convertSet : mastersStatus[m.here].status = \text{“convertDead”}
\end{array}
$$

$$
\begin{array}{l}
ConvertDeadActivities \triangleq \\
\quad \land\ pstate = \text{“running”} \\
\quad \land\ \exists\, p \in PLACE : mastersStatus[p].status = \text{“convertDead”} \\
\quad \land\ \text{LET}\ convSeeker \triangleq GetConvertSeeker \\
\qquad \text{IN}\ \ \land\ convSeeker \neq C!NotConvTask \\
\qquad\qquad \land\ convSeeker.here \notin killed \\
\qquad\qquad \land\ \text{LET}\ here \triangleq convSeeker.here \\
\qquad\qquad\qquad\quad pl \triangleq convSeeker.pl \\
\qquad\qquad\qquad\quad fid \triangleq convSeeker.fid \\
\qquad\qquad\qquad\quad dead \triangleq mastersStatus[here].lastKilled \\
\qquad\qquad\quad \text{IN}\ \ \land\ SetActionNameAndDepth(\langle \text{“ConvertDeadActivities”}, here\rangle) \\
\qquad\qquad\qquad\quad \land\ convertSet' = convertSet \setminus \{convSeeker\} \\
\qquad\qquad\qquad\quad \land\ fmasters[fid].transitAdopted[pl][dead] \geq 0 \\
\qquad\qquad\qquad\quad \land\ fmasters[fid].transitAdopted[dead][pl] \geq 0 \\
\qquad\qquad\qquad\quad \land\ fmasters[fid].liveAdopted[dead] \geq 0 \\
\qquad\qquad\qquad\quad \land\ fmasters' = [fmasters\ \text{EXCEPT}\ ![fid].numActive = \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad @ - fmasters[fid].transit[pl][dead] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad - fmasters[fid].transit[dead][pl] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad - fmasters[fid].transitAdopted[pl][dead] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad - fmasters[fid].transitAdopted[dead][pl] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad - fmasters[fid].live[dead] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad - fmasters[fid].liveAdopted[dead], \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad ![fid].transit[pl][dead] = 0,
\end{array}
$$

```
                                              ![fid].transitAdopted[pl][dead]  = 0,
                                              ![fid].transit[dead][pl]  = 0,
                                              ![fid].transitAdopted[dead][pl]  = 0,
                                              ![fid].live[dead] = 0,
                                              ![fid].liveAdopted[dead] = 0]
                         ∧ IF fmasters'[fid].numActive = 0
                             THEN ∧ C!SendMsg([mid  ↦ seq.mseq,
                                               src  ↦ here,
                                               dst  ↦ here,
                                               fid  ↦ fid,
                                               type ↦ "releaseFinish"])
                                  ∧ C!IncrMSEQ(1)
                             ELSE ∧ msgs' = msgs
                                  ∧ seq' = seq
                         ∧ IF  ∃ m ∈ convertSet' : m.here = here
                             THEN  mastersStatus' = mastersStatus
                             ELSE  mastersStatus' = [mastersStatus EXCEPT ![here].status = "running"]
  ∧ UNCHANGED ⟨fstates, pstate, thrds, killed, pendingAct,  fbackups, waitForMsgs,
               adoptSet,  blockedThrds, runningThrds⟩
```

```
FindWaitForMSG ≜
  LET mset ≜ {m ∈ waitForMsgs :
                ∧ m.src  ∈ killed
                ∧ m.dst  ∉ killed
                ∧ m.src ∈ killed}
  IN   IF mset = {} THEN C!NotMessage
       ELSE  CHOOSE x ∈ mset : TRUE

SimulateFailedResponse ≜
  ∧ pstate = "running"
  ∧ killed ≠ {}
  ∧ waitForMsgs ≠ {}
  ∧ LET msg ≜ FindWaitForMSG
    IN     ∧ msg ≠ C!NotMessage
           ∧ LET dead ≜ msg.src
                 here ≜ msg.dst
                 delMsgs ≜ {m ∈ msgs : m.dst = dead }
                  wfm ≜ {m ∈ waitForMsgs : m.dst = dead}
             IN    ∧ SetActionNameAndDepth(⟨"SimulateFailedResponse", here⟩)
                   ∧ waitForMsgs' = (waitForMsgs \ wfm) \ {msg}
                   ∧ C!IncrMSEQ(1)
                   ∧ IF msg.type = "masterLiveDone"
                       THEN IF  ¬(∃ m ∈ msgs :  message has been sent already
                                     ∧ m.type = msg.type ∧ m.src = msg.src
```

$$
\begin{array}{l}
\qquad\qquad \wedge\, m.dst = msg.dst \wedge m.fid = msg.fid \\
\qquad\qquad \wedge\, m.aid = msg.aid \wedge m.success) \\
\text{THEN} \quad \wedge\, msgs' = (msgs \setminus delMsgs) \cup \{ \\
\qquad\qquad\qquad [\ mid \mapsto seq.mseq, \\
\qquad\qquad\qquad\ \ src \mapsto msg.src, \\
\qquad\qquad\qquad\ \ dst \mapsto msg.dst, \\
\qquad\qquad\qquad\ source \mapsto msg.source, \\
\qquad\qquad\qquad\ target \mapsto msg.target, \\
\qquad\qquad\qquad\ \ fid \mapsto msg.fid, \\
\qquad\qquad\qquad\ \ aid \mapsto msg.aid, \\
\qquad\qquad\qquad\ type \mapsto \text{“masterLiveDone”}, \\
\qquad\qquad\qquad\ submit \mapsto \text{FALSE}, \\
\qquad\qquad\qquad\ success \mapsto \text{FALSE}, \\
\qquad\qquad\qquad\ isAdopter \mapsto \text{FALSE}, \\
\qquad\qquad\qquad\ adoptedFID \mapsto C!NotID, \\
\qquad\qquad\qquad\ backupPlace \mapsto C!NotPlace]\} \\
\qquad\quad \text{ELSE} \quad \wedge\, msgs' = (msgs \setminus delMsgs) \\
\text{ELSE IF}\ msg.type = \text{“masterCompletedDone”} \\
\text{THEN IF} \quad \neg(\exists\, m \quad \in msgs : \ \text{message has been sent already} \\
\qquad\qquad \wedge\, m.type = msg.type \wedge m.src = msg.src \\
\qquad\qquad \wedge\, m.dst = msg.dst \wedge m.fid = msg.fid \\
\qquad\qquad \wedge\, m.isAdopter = msg.isAdopter \\
\qquad\qquad \wedge\, m.success) \\
\qquad\quad \text{THEN} \quad \wedge\, msgs' = (msgs \setminus delMsgs) \cup \{ \\
\qquad\qquad\qquad [\ mid \mapsto seq.mseq, \\
\qquad\qquad\qquad\ \ src \mapsto msg.src, \\
\qquad\qquad\qquad\ \ dst \mapsto msg.dst, \\
\qquad\qquad\qquad\ target \mapsto msg.target, \\
\qquad\qquad\qquad\ \ fid \mapsto msg.fid, \\
\qquad\qquad\qquad\ type \mapsto \text{“masterCompletedDone”}, \\
\qquad\qquad\qquad\ success \mapsto \text{FALSE}, \\
\qquad\qquad\qquad\ isAdopter \mapsto \text{FALSE}, \\
\qquad\qquad\qquad\ finishEnd \mapsto \text{FALSE}, \\
\qquad\qquad\qquad\ backupPlace \mapsto C!NotPlace]\} \\
\qquad\quad \text{ELSE} \quad \wedge\, msgs' = (msgs \setminus delMsgs) \\
\text{ELSE IF}\ msg.type = \text{“masterTransitDone”} \\
\text{THEN IF}\ \neg(\exists\, m \in msgs : \ \text{message has been sent already} \\
\qquad\qquad \wedge\, m.type = msg.type \wedge m.src = msg.src \\
\qquad\qquad \wedge\, m.dst = msg.dst \wedge m.fid = msg.fid \\
\qquad\qquad \wedge\, m.success) \\
\qquad\quad \text{THEN} \quad \wedge\, msgs' = (msgs \setminus delMsgs) \cup \{ \\
\qquad\qquad\qquad [\ mid \mapsto seq.mseq, \\
\qquad\qquad\qquad\ \ src \mapsto msg.src, \\
\qquad\qquad\qquad\ \ dst \mapsto msg.dst, \\
\qquad\qquad\qquad\ target \mapsto msg.target,
\end{array}
$$

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{fid} \mapsto \mathit{msg.fid}, \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{type} \mapsto \text{“masterTransitDone”}, \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{isAdopter} \mapsto \text{FALSE}, \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{adoptedFID} \mapsto C!\mathit{NotID}, \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{backupPlace} \mapsto C!\mathit{NotPlace}, \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{submit} \mapsto \text{FALSE}, \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{success} \mapsto \text{FALSE}]\} \\
\qquad \text{ELSE} \quad \land\ \mathit{msgs}' = (\mathit{msgs} \setminus \mathit{delMsgs}) \\
\text{ELSE IF } \mathit{msg.type} = \text{“backupCompletedDone”} \\
\text{THEN IF } \neg(\exists\, m \in \mathit{msgs} : \quad \text{message has been sent already} \\
\qquad\qquad\qquad \land\ m.\mathit{type} = \mathit{msg.type} \land m.\mathit{src} = \mathit{msg.src} \\
\qquad\qquad\qquad \land\ m.\mathit{dst} = \mathit{msg.dst} \land m.\mathit{fid} = \mathit{msg.fid} \\
\qquad\qquad\qquad \land\ m.\mathit{isAdopter} = \mathit{msg.isAdopter} \land m.\mathit{success}) \\
\qquad \text{THEN} \quad \land\ \mathit{msgs}' = (\mathit{msgs} \setminus \mathit{delMsgs}) \cup \{ \\
\qquad\qquad\qquad\qquad\qquad [\ \mathit{mid} \mapsto \mathit{seq.mseq}, \\
\qquad\qquad\qquad\qquad\qquad\quad \mathit{src} \mapsto \mathit{msg.src}, \\
\qquad\qquad\qquad\qquad\qquad\quad \mathit{dst} \mapsto \mathit{msg.dst}, \\
\qquad\qquad\qquad\qquad\qquad\quad \mathit{target} \mapsto \mathit{msg.target}, \\
\qquad\qquad\qquad\qquad\qquad\quad \mathit{fid} \mapsto \mathit{msg.fid}, \\
\qquad\qquad\qquad\qquad\qquad\quad \mathit{type} \mapsto \text{“backupCompletedDone”}, \\
\qquad\qquad\qquad\qquad\qquad\quad \mathit{isAdopter} \mapsto \mathit{msg.isAdopter}, \\
\qquad\qquad\qquad\qquad\qquad\quad \mathit{success} \mapsto \text{FALSE}]\} \\
\qquad \text{ELSE} \quad \land\ \mathit{msgs}' = (\mathit{msgs} \setminus \mathit{delMsgs}) \\
\text{ELSE IF } \mathit{msg.type} = \text{“backupLiveDone”} \\
\text{THEN IF } \neg(\exists\, m \in \mathit{msgs} : \quad \text{message has been sent already} \\
\qquad\qquad\qquad \land\ m.\mathit{type} = \mathit{msg.type} \land m.\mathit{src} = \mathit{msg.src} \\
\qquad\qquad\qquad \land\ m.\mathit{dst} = \mathit{msg.dst} \land m.\mathit{fid} = \mathit{msg.fid} \\
\qquad\qquad\qquad \land\ m.\mathit{source} = \mathit{msg.source} \land m.\mathit{success}) \\
\qquad \text{THEN} \quad \land\ \mathit{msgs}' = (\mathit{msgs} \setminus \mathit{delMsgs}) \cup \{ \\
\qquad\qquad\qquad\qquad\qquad [\ \mathit{mid} \mapsto \mathit{seq.mseq}, \\
\qquad\qquad\qquad\qquad\qquad\quad \mathit{src} \mapsto \mathit{msg.src}, \\
\qquad\qquad\qquad\qquad\qquad\quad \mathit{dst} \mapsto \mathit{msg.dst}, \\
\qquad\qquad\qquad\qquad\qquad\quad \mathit{target} \mapsto \mathit{msg.target}, \\
\qquad\qquad\qquad\qquad\qquad\quad \mathit{source} \mapsto \mathit{msg.source}, \\
\qquad\qquad\qquad\qquad\qquad\quad \mathit{fid} \mapsto \mathit{msg.fid}, \\
\qquad\qquad\qquad\qquad\qquad\quad \mathit{aid} \mapsto \mathit{msg.aid}, \\
\qquad\qquad\qquad\qquad\qquad\quad \mathit{type} \mapsto \text{“backupLiveDone”}, \\
\qquad\qquad\qquad\qquad\qquad\quad \mathit{isAdopter} \mapsto \mathit{msg.isAdopter}, \\
\qquad\qquad\qquad\qquad\qquad\quad \mathit{adoptedFID} \mapsto \mathit{msg.adoptedFID}, \\
\qquad\qquad\qquad\qquad\qquad\quad \mathit{success} \mapsto \text{FALSE}]\} \\
\qquad \text{ELSE} \quad \land\ \mathit{msgs}' = (\mathit{msgs} \setminus \mathit{delMsgs}) \\
\text{ELSE IF } \mathit{msg.type} = \text{“backupTransitDone”} \\
\text{THEN IF } \neg(\exists\, m \in \mathit{msgs} : \quad \text{message has been sent already} \\
\qquad\qquad\qquad \land\ m.\mathit{type} = \mathit{msg.type} \land m.\mathit{src} = \mathit{msg.src} \\
\qquad\qquad\qquad \land\ m.\mathit{dst} = \mathit{msg.dst} \land m.\mathit{fid} = \mathit{msg.fid}
\end{array}
$$

$\wedge\ m.target = msg.target \wedge m.success)$

$\textsc{then}\ \wedge\ msgs' = (msgs \setminus delMsgs) \cup \{$

$[\ mid \mapsto seq.mseq,$
$src \mapsto msg.src,$
$dst \mapsto msg.dst,$
$target \mapsto msg.target,$
$fid \mapsto msg.fid,$
$type \mapsto$ "backupTransitDone",
$isAdopter \mapsto msg.isAdopter,$
$adoptedFID \mapsto msg.adoptedFID,$
$success \mapsto \text{FALSE}]\}$

$\textsc{else}\ \wedge\ msgs' = (msgs \setminus delMsgs)$

$\textsc{else}\ \text{FALSE}$

$\wedge\ \textsc{unchanged}\ \langle convertSet, adoptSet, mastersStatus, fstates, pstate,$
$thrds, killed, pendingAct, fmasters, fbackups,$
$blockedThrds, runningThrds\rangle$

Predicate enumerating all possible next actions

$Next \triangleq$

- $\vee\ RecvAsync$
- $\vee\ ReleaseRootFinish$
- $\vee\ AuthorizeReceivedAsync$
- $\vee\ BackupTransit$
- $\vee\ BackupLive$
- $\vee\ BackupCompleted$
- $\vee\ BackupGetAdopter$
- $\vee\ MasterTransit$
- $\vee\ MasterLive$
- $\vee\ MasterCompleted$
- $\vee\ MasterTransitDone$
- $\vee\ MasterLiveDone$
- $\vee\ MasterCompletedDone$
- $\vee\ AdopterTransit$
- $\vee\ AdopterLive$
- $\vee\ AdopterCompleted$
- $\vee\ SeekAdoption$
- $\vee\ ConvertDeadActivities$
- $\vee\ SimulateFailedResponse$
- $\vee\ GetAdopterDone$
- $\vee\ RunExprOrKill$
- $\vee\ ScheduleNestedFinish$
- $\vee\ TerminateAsync$
- $\vee\ SpawnRemoteAsync$
- $\vee\ SpawnLocalAsync$
- $\vee\ StopFinish$

$\lor$ *StartFinish*
$\lor$ *AuthorizeTransitAsync*
$\lor$ *UnblockTerminateAsync*

Asserting fairness properties to all actions

Liveness $\triangleq$
$\land \mathrm{WF}_{Vars}(RecvAsync)$
$\land \mathrm{WF}_{Vars}(ReleaseRootFinish)$
$\land \mathrm{WF}_{Vars}(AuthorizeReceivedAsync)$
$\land \mathrm{WF}_{Vars}(StartFinish)$
$\land \mathrm{WF}_{Vars}(StopFinish)$
$\land \mathrm{WF}_{Vars}(SpawnLocalAsync)$
$\land \mathrm{WF}_{Vars}(SpawnRemoteAsync)$
$\land \mathrm{WF}_{Vars}(TerminateAsync)$
$\land \mathrm{WF}_{Vars}(ScheduleNestedFinish)$
$\land \mathrm{WF}_{Vars}(RunExprOrKill)$
$\land \mathrm{WF}_{Vars}(BackupTransit)$
$\land \mathrm{WF}_{Vars}(BackupLive)$
$\land \mathrm{WF}_{Vars}(BackupCompleted)$
$\land \mathrm{WF}_{Vars}(MasterTransit)$
$\land \mathrm{WF}_{Vars}(MasterLive)$
$\land \mathrm{WF}_{Vars}(MasterCompleted)$
$\land \mathrm{WF}_{Vars}(MasterTransitDone)$
$\land \mathrm{WF}_{Vars}(MasterLiveDone)$
$\land \mathrm{WF}_{Vars}(MasterCompletedDone)$
$\land \mathrm{WF}_{Vars}(AdopterTransit)$
$\land \mathrm{WF}_{Vars}(AdopterLive)$
$\land \mathrm{WF}_{Vars}(AdopterCompleted)$
$\land \mathrm{WF}_{Vars}(SeekAdoption)$
$\land \mathrm{WF}_{Vars}(ConvertDeadActivities)$
$\land \mathrm{WF}_{Vars}(SimulateFailedResponse)$
$\land \mathrm{WF}_{Vars}(GetAdopterDone)$
$\land \mathrm{WF}_{Vars}(BackupGetAdopter)$
$\land \mathrm{WF}_{Vars}(AuthorizeTransitAsync)$
$\land \mathrm{WF}_{Vars}(UnblockTerminateAsync)$

Specification

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

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

* Modification History
* Last modified *Fri Dec* 15 11:49:46 *AEDT* 2017 by *u5482878*
* Last modified Sun *Dec* 10 18:15:04 *AEDT* 2017 by *shamouda*

\ * Created *Wed Sep* 13 12:14:43 *AEST* 2017 by *u*5482878