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 as a sequence of *async* statements
MXFINISHES,	Maximum finish objects including root and remote
BACKUP,	A function from place to its backup
DEPTH	Maximum expected depth of the trance

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

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

Predicate to hide the finish implementation

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

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

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

$$\begin{aligned} P0 &\triangleq PROG\_HOME\\ P1 &\triangleq BACKUP[PROG\_HOME]\\ P2 &\triangleq BACKUP[BACKUP[PROG\_HOME]] \end{aligned}$$

Invariants (formulas true in every reachable state.)

$$\begin{aligned} &TypeOK \triangleq\\ &\quad \wedge fstates \in [C!IDRange \to C!FinishState]\\ &\quad \wedge thrds \in [PLACE \to [\ C!ThreadID \to C!Thread]]\\ &\quad \wedge msgs \subseteq C!Messages\\ &\quad \wedge pstate \in \{\text{“running”},\ \text{“terminated”},\ \text{“exceptionThrown”}\} \end{aligned}$$

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

$StateOK\ \triangleq$ TRUE

$MustTerminate\ \triangleq$
$\Diamond(pstate = \text{“terminated”})$

Initialization

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

$$
\begin{array}{l}
\qquad\qquad\quad adoptedRoot \mapsto C!NotID, \\
\qquad\qquad\quad numActive \mapsto 0, \\
\qquad\qquad\quad isReleased \mapsto \text{FALSE}]] \\
\land\ pstate = \text{“running”} \\
\land\ mastersStatus = [p \in PLACE \mapsto [\ status \mapsto \text{“running”}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad lastKilled \mapsto C!NotPlace]] \\
\land\ msgs = \{\} \\
\land\ seq = [aseq \mapsto 1,\ fseq \mapsto C!FIRST\_ID,\ mseq \mapsto 1] \\
\land\ thrds = [p \in PLACE \mapsto \\
\qquad [t \in C!ThreadID \mapsto \\
\qquad\quad \text{IF } p = PROG\_HOME \land t = 0 \\
\qquad\quad \text{THEN } [tid \mapsto t,\ status \mapsto \text{“running”}, \\
\qquad\qquad\quad blockingType \mapsto \text{“NA”}, \\
\qquad\qquad\quad stack \mapsto \langle [\ b \mapsto 0, \\
\qquad\qquad\qquad\qquad\quad i \mapsto \text{IF } PROG[0].type = \text{“finish”} \\
\qquad\qquad\qquad\qquad\qquad\quad \text{THEN } C!I\_PRE\_FIN\_ALLOC \\
\qquad\qquad\qquad\qquad\qquad\quad \text{ELSE } C!I\_START, \\
\qquad\qquad\qquad\qquad fid \mapsto C!NoParent]\rangle] \\
\qquad\quad \text{ELSE } [tid \mapsto t,\ status \mapsto \text{“idle”}, \\
\qquad\qquad\quad blockingType \mapsto \text{“NA”}, \\
\qquad\qquad\quad stack \mapsto \langle\rangle]]] \\
\land\ killed = \{\} \\
\land\ pendingAct = \{\} \\
\land\ waitForMsgs = \{\} \\
\land\ runningThrds = \{[here \mapsto PROG\_HOME,\ tid \mapsto 0]\} \\
\land\ blockedThrds = \{\} \\
\land\ adoptSet = \{\} \\
\land\ convertSet = \{\}
\end{array}
$$

Helper Actions

$$
\begin{array}{l}
SetActionNameAndDepth(name) \triangleq \\
\quad \text{IF } depth = DEPTH \text{ THEN TRUE ELSE } \land actionName' = name \land depth' = depth + 1
\end{array}
$$

$$
\begin{array}{l}
FindPendingActivity(actId) \triangleq \\
\quad \text{LET } aset \triangleq \{a \in pendingAct : a.aid = actId\} \\
\quad \text{IN } \text{IF } aset = \{\} \text{ THEN } C!NotActivity \\
\qquad\quad \text{ELSE CHOOSE } x \in aset : \text{TRUE}
\end{array}
$$

$$
\begin{array}{l}
FindIdleThread(here) \triangleq \\
\quad \text{LET } idleThreads \triangleq C!PlaceThread \setminus (runningThrds \cup blockedThrds) \\
\qquad tset \triangleq \{t \in idleThreads : \\
\qquad\qquad\qquad \land\ t.here = here \\
\qquad\qquad\qquad \land\ t.here \notin killed \\
\qquad\qquad\qquad \land\ thrds[t.here][t.tid].status = \text{“idle”}\} \\
\quad \text{IN } \text{IF } tset = \{\} \text{ THEN } C!NotPlaceThread
\end{array}
$$

```
        ELSE  CHOOSE x ∈ 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}]
```

$\land\ fbackups' = [fbackups \text{ EXCEPT } ![encRoot].children = @ \cup \{newFid\}]$

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

$FindRunningThreadForScheduleNestedFinish \triangleq$
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 $\land\ PROG[blk].type \notin \{\text{“expr”}, \text{“kill”}\}$
$\land\ curStmt \geq 0$
$\land\ curStmt \leq PROG[blk].mxstmt$
$\land\ PROG[nested].type = \text{“finish”}$
$\land\ PROG[nested].dst = t.here\ \}$
IN IF $tset = \{\}$ THEN $C!NotPlaceThread$
ELSE CHOOSE $x \in tset$: TRUE

Processing a nested finish in the currently running block

$ScheduleNestedFinish \triangleq$
$\land\ pstate = \text{“running”}$
$\land$ LET $pthrd \triangleq FindRunningThreadForScheduleNestedFinish$
IN $\land\ pthrd \neq C!NotPlaceThread$
$\land$ LET $here \triangleq pthrd.here$
$tid \triangleq pthrd.tid$
$top \triangleq Head(thrds[here][tid].stack)$
$tail \triangleq Tail(thrds[here][tid].stack)$
$lstStmt \triangleq top.i$
$curStmt \triangleq top.i + 1$
$blk \triangleq top.b$
$fid \triangleq top.fid$
$nested \triangleq PROG[blk].stmts[curStmt]$
$newFid \triangleq seq.fseq$
$encRoot \triangleq C!GetEnclosingRoot(fid, newFid)$
IN $\land\ SetActionNameAndDepth(\langle \text{“ScheduleNestedFinish”}, here\rangle)$
$\land\ thrds' = [thrds \text{ EXCEPT } ![here][tid].stack =$
$\langle[\ b \mapsto nested,$
$i \mapsto C!I\_START,$
$fid \mapsto newFid],$
$[\ b \mapsto top.b,$
$i \mapsto curStmt,$

```
                                                    fid ↦ fid]
                                                ⟩ ∘ tail]
                    ∧ Finish(seq.fseq)!Alloc(C!ROOT_FINISH, here, fid, newFid)
                    ∧ C!IncrFSEQ
                    ∧ fmasters′ = [fmasters EXCEPT ![encRoot].children =
                                                    @ ∪ {newFid}]
                    ∧ fbackups′ = [fbackups EXCEPT ![encRoot].children =
                                                    @ ∪ {newFid}]
  ∧ UNCHANGED ⟨convertSet, adoptSet, mastersStatus, msgs, pstate, waitForMsgs,
               killed, pendingAct, runningThrds, blockedThrds⟩
```

```
FindRunningThreadForSpawnLocalAsync ≜
  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 = "async"
                       ∧ PROG[nested].dst = t.here
              }
  IN  IF tset = {} THEN C!NotPlaceThread
      ELSE CHOOSE x ∈ tset : TRUE
```

Processing a nested local *async* in the currently running block

```
SpawnLocalAsync ≜
  ∧ pstate = "running"
  ∧ LET pthrd ≜ FindRunningThreadForSpawnLocalAsync
    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]
                idle ≜ FindIdleThread(here)
                act ≜ [aid ↦ seq.aseq, b ↦ nested,  fid ↦ fid]
```

```
                stkEntry ≜ [b ↦ act.b, i ↦ C!I_START, fid ↦ act.fid]
        IN      ∧ SetActionNameAndDepth(⟨"SpawnLocalAsync", here⟩)
                ∧ IF act.fid ≠ C!NoParent
                    THEN Finish(act.fid)!NotifyLocalActivitySpawnAndCreation(here, act)
                    ELSE fstates' = fstates
                ∧ C!IncrASEQ
                ∧ thrds' = [thrds EXCEPT ![here][tid].stack =
                                                ⟨[ b   ↦ top.b,
                                                   i   ↦ curStmt,
                                                   fid ↦ fid]
                                                ⟩ ∘ tail,
                                           ![here][idle.tid].stack = ⟨stkEntry⟩,
                                           ![here][idle.tid].status = "running"]
                ∧ runningThrds' = runningThrds ∪ {[here ↦ here, tid ↦ idle.tid]}
    ∧ UNCHANGED ⟨convertSet, adoptSet, mastersStatus, msgs, pstate, killed,
                 pendingAct, fmasters, fbackups, waitForMsgs, blockedThrds⟩

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

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

Processing a nested remote async in the currently running block
SpawnRemoteAsync ≜
  ∧ pstate = "running"
  ∧ LET pthrd ≜ FindRunningThreadForSpawnRemoteAsync
    IN  ∧ pthrd ≠ C!NotPlaceThread
        ∧ LET here ≜ pthrd.here
              tid ≜ pthrd.tid
              top ≜ Head(thrds[here][tid].stack)
```

$tail \triangleq Tail(thrds[here][tid].stack)$
$lstStmt \triangleq top.i$
$curStmt \triangleq top.i + 1$
$blk \triangleq top.b$
$fid \triangleq top.fid$
$root \triangleq GetRootFinishId(fid)$
$nested \triangleq PROG[blk].stmts[curStmt]$
$dst \triangleq PROG[nested].dst$
IN $\land SetActionNameAndDepth(\langle$"SpawnRemoteAsync", $here$, "to", $dst\rangle)$
$\land Finish(fid)!NotifySubActivitySpawn(dst)$
$\land thrds' = [thrds$ EXCEPT $![here][tid].status =$ "blocked",
$![here][tid].blockingType =$ "AsyncTransit"]
$\land blockedThrds' = blockedThrds \cup \{[here \mapsto here, tid \mapsto tid]\}$
$\land runningThrds' = runningThrds \setminus \{[here \mapsto here, tid \mapsto tid]\}$
$\land$ UNCHANGED $\langle convertSet, adoptSet, mastersStatus, pstate, killed, pendingAct, fmasters, fbackups\rangle$

$FindRunningThreadForRunExprOrKill \triangleq$
LET $tset \triangleq \{t \in runningThrds :$
$\land t.here \notin killed$
$\land thrds[t.here][t.tid].status =$ "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 $\land PROG[blk].type \notin \{$"expr", "kill"$\}$
$\land curStmt \geq 0$
$\land curStmt \leq PROG[blk].mxstmt$
$\land PROG[nested].type \in \{$"expr", "kill"$\}$ $\}$
IN IF $tset = \{\}$ THEN $C!NotPlaceThread$
ELSE CHOOSE $x \in tset$: TRUE

$Kill(dead) \triangleq$
$\land killed' = killed \cup \{dead\}$
$\land adoptSet' = adoptSet \cup \{m \in C!Adopter :$
$\land m.child \neq C!NotID$
$\land m.adopter \neq C!NotID$
$\land m.here \neq dead$
$\land m.here = fstates[m.adopter].here$
$\land m.child \in fmasters[m.adopter].children$
$\land fbackups[m.child].isAdopted =$ FALSE
$\land fstates[m.child].here = dead$
$\land m.adopter = fstates[m.child].eroot\}$
$\land$ IF $adoptSet' = \{\}$

```
        THEN ∧ mastersStatus' = [mastersStatus EXCEPT ![PROG_HOME].status = "convertDead",
                                                      ![PROG_HOME].lastKilled = dead]
             ∧ convertSet' =  convertSet ∪ {t ∈ C!ConvTask :
                                            ∧ t.pl ≠ C!NotPlace
                                            ∧ t.pl ≠ dead
                                            ∧ t.pl ∉ killed
                                            ∧ t.fid = C!FIRST_ID
                                            ∧ t.here = PROG_HOME}
        ELSE ∧ mastersStatus' = [p ∈ PLACE ↦ IF ∃ m ∈ adoptSet' : m.here = p
                                              THEN [    status   ↦ "seekAdoption",
                                                     lastKilled ↦ dead]
                                              ELSE [    status   ↦ "running",
                                                     lastKilled ↦ C!NotPlace]]
             ∧ convertSet' =  convertSet
   ∧ 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
```

$\land\ msgs' = msgs$
$\land\ waitForMsgs' = waitForMsgs$
ELSE $\land\ Kill(PROG[nested].dst)$
$\land$ UNCHANGED $\langle fstates,\ pstate,\ seq,\ pendingAct,\ fmasters,\ fbackups,$
$runningThrds,\ blockedThrds\rangle$

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

Running thread processing the end of an *async* block
$TerminateAsync \triangleq$
$\land\ pstate =$ "running"
$\land$ LET $pthrd \triangleq FindRunningThreadForTerminateAsync$
IN $\land\ pthrd \neq C!NotPlaceThread$
$\land$ LET $here \triangleq pthrd.here$
$tid \triangleq pthrd.tid$
$top \triangleq Head(thrds[here][tid].stack)$
$blk \triangleq top.b$
$fid \triangleq top.fid$
IN $\land\ SetActionNameAndDepth(\langle$"TerminateAsync", $here\rangle)$
$\land\ Finish(fid)!NotifyActivityTermination($FALSE$)$
$\land\ thrds' = [thrds$ EXCEPT $![here][tid].status =$ "blocked",
$![here][tid].blockingType =$ "AsyncTerm"$]$
$\land\ runningThrds' = runningThrds \setminus \{[here \mapsto here,\ tid \mapsto tid]\}$
$\land\ blockedThrds' = blockedThrds \cup \{[here \mapsto here,\ tid \mapsto tid]\}$
$\land$ UNCHANGED $\langle convertSet,\ adoptSet,\ mastersStatus,\ pstate,\ killed,$
$pendingAct,\ fmasters,\ fbackups\rangle$

$FindRunningThreadForStopFinish \triangleq$
LET $tset \triangleq \{t \in runningThrds :$
$\land\ t.here \notin killed$
$\land\ thrds[t.here][t.tid].status =$ "running"
$\land$ LET $top \triangleq Head(thrds[t.here][t.tid].stack)$
IN $\land\ PROG[top.b].type =$ "finish"
$\land\ PROG[top.b].mxstmt = top.i\ \}$
IN IF $tset = \{\}$ THEN $C!NotPlaceThread$

```
        ELSE  CHOOSE x ∈ tset : TRUE

 Running thread processing the end of a finish block and blocking itself
StopFinish ≜
  ∧ pstate = "running"
  ∧ LET pthrd ≜ FindRunningThreadForStopFinish
    IN  ∧ pthrd ≠ C!NotPlaceThread
        ∧ LET here ≜ pthrd.here
              tid  ≜ pthrd.tid
               top ≜ Head(thrds[here][tid].stack)
          IN  ∧ SetActionNameAndDepth(⟨"StopFinish", here⟩)
              ∧ PROG[top.b].type = "finish"
              ∧ PROG[top.b].mxstmt = top.i
              ∧ Finish(top.fid)!NotifyActivityTermination(TRUE)
              ∧ thrds' = [thrds EXCEPT ![here][tid].status = "blocked",
                                       ![here][tid].blockingType = "FinishEnd"]
              ∧ runningThrds' = runningThrds \ {[here ↦ here, tid ↦ tid]}
              ∧ blockedThrds' = blockedThrds ∪ {[here ↦ here, tid ↦ tid]}
  ∧ UNCHANGED ⟨convertSet, adoptSet, mastersStatus, pstate, killed, pendingAct,
               fmasters, fbackups⟩

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

RecvAsync ≜
  ∧ pstate = "running"
  ∧ LET msg ≜ C!FindMSG("async")
    IN  ∧ msg ≠ C!NotMessage
        ∧ LET here ≜ msg.dst
              pid  ≜ msg.fid
              fid  ≜ C!GetActiveFID(C!REMOTE_FINISH, here, pid)
              src  ≜ msg.src
              blk  ≜ msg.b
              newFID ≜ seq.fseq
              activity ≜ [aid ↦ seq.aseq, b ↦ blk, fid ↦ newFID]
          IN  ∧ SetActionNameAndDepth(⟨"RecvAsync", here⟩)
              ∧ pid ≠ C!NotID
              ∧ fid  = C!NotID   we don't reuse remote finishes
              ∧ src ≠ C!NotPlace
              ∧ Finish(activity.fid)!AllocRemoteAndNotifyRemoteActivityCreation(
                                     src, activity, msg, C!REMOTE_FINISH,
                                     here,  parentpid,  rootpid)
              ∧ pendingAct' = pendingAct ∪ {activity}
              ∧ C!IncrAll
  ∧ UNCHANGED ⟨convertSet, adoptSet, mastersStatus, pstate, thrds,
               killed,  fmasters, fbackups, blockedThrds, runningThrds⟩

──────────────────────────────────────────────────────────────────────────
```

```
FindBlockedThreadMasterTransitDone ≜
    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, "masterTransitDone") ≠ C!NotMessage }
    IN  IF tset = {} THEN C!NotPlaceThread
        ELSE CHOOSE x ∈ tset : TRUE

MasterTransitDone ≜
  ∧ pstate = "running"
  ∧ msgs ≠ {}
  ∧ LET pthrd ≜ FindBlockedThreadMasterTransitDone
    IN  ∧ pthrd ≠ C!NotPlaceThread
        ∧ LET here ≜ pthrd.here
              tid ≜ pthrd.tid
              msg ≜ C!FindIncomingMSG(here, "masterTransitDone")
              success ≜ msg.success
              submit ≜ msg.submit
              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)
              nested ≜ PROG[blk].stmts[curStmt]
              asyncDst ≜ PROG[nested].dst
              isAdopter ≜ msg.isAdopter
              backupPlace ≜ msg.backupPlace
              adoptedFID ≜ msg.adoptedFID
              masterWFM ≜ [src ↦ rootPlace,
                           dst ↦ here,
                           fid ↦ root,
                        target ↦ asyncDst,
                          type ↦ "masterTransitDone" ]
              backupWFM ≜ [src ↦ backupPlace,
                           dst ↦ here,
                           fid ↦ root,
                        target ↦ asyncDst,
                     isAdopter ↦ isAdopter,
                   adoptedFID ↦ adoptedFID,
                          type ↦ "backupTransitDone" ]
          IN  ∧ SetActionNameAndDepth(⟨"MasterTransitDone", here,
```

```
                                        "success", success,
                                        "submit", submit⟩)
∧ IF success ∧ submit ∧ rootPlace ∉ killed
  THEN ∧ C!ReplaceMsg(msg, [ mid    ↦ seq.mseq,
                             src    ↦ here,
                             dst    ↦ backupPlace,
                             target ↦ asyncDst,
                             fid    ↦ root,
                             isAdopter ↦ isAdopter,
                             adoptedFID ↦ adoptedFID,
                             type   ↦ "backupTransit"])
       ∧ thrds' = thrds
       ∧ blockedThrds'  = blockedThrds
       ∧ runningThrds' = runningThrds
       ∧ waitForMsgs' = (waitForMsgs \ {masterWFM}) ∪ {backupWFM}
       ∧ C!IncrMSEQ(1)
  ELSE IF success ∧ rootPlace ∉ killed  ignore the async, go to the next step
  THEN ∧ C!RecvMsg(msg)
       ∧ 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 \ {masterWFM}
       ∧ C!IncrMSEQ(1)
  ELSE ∧ C!ReplaceMsg(msg, [ mid    ↦ seq.mseq,
                             src    ↦ here,
                             dst    ↦ C!GetBackup(rootPlace),
                             source ↦ here,
                             target ↦ asyncDst,
                             fid    ↦ root,
                             type   ↦ "backupGetAdopter",
                             actionType ↦ "transit",
                             aid    ↦ C!NotActivity.aid,
                             finishEnd ↦ FALSE])
       ∧ thrds' = thrds
       ∧ blockedThrds'  = blockedThrds
       ∧ runningThrds' = runningThrds
       ∧ waitForMsgs' = waitForMsgs \ {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 $\wedge\ SetActionNameAndDepth(\langle \text{“MasterLiveDone”}, here\rangle)$
$\wedge\ activity \neq C!NotActivity$
$\wedge\ fstates[activity.fid].here = here$
$\wedge$ IF $success \wedge submit \wedge rootPlace \notin killed$
THEN $\wedge\ C!ReplaceMsg(msg, [\ \ mid \mapsto seq.mseq,$
$src\ \mapsto here,$

$dst \mapsto backupPlace,$
$source \mapsto source,$
$target \mapsto here,$
$fid \mapsto root,$
$aid \mapsto actId,$
$type \mapsto$ "backupLive",
$isAdopter \mapsto isAdopter,$
$adoptedFID \mapsto adoptedFID])$

$\land\ waitForMsgs' = (waitForMsgs \setminus \{masterWFM\}) \cup \{backupWFM\}$
$\land\ C!IncrMSEQ(1)$
$\land\ pendingAct' = pendingAct$

ELSE IF $success \land rootPlace \notin killed$
THEN $\land\ C!RecvMsg(msg)$
$\land\ pendingAct' = pendingAct \setminus \{activity\}$
$\land\ seq' = seq$
$\land\ waitForMsgs' = waitForMsgs \setminus \{masterWFM\}$

ELSE $\land\ C!ReplaceMsg(msg, [\ mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto C!GetBackup(rootPlace),$
$source \mapsto source,$
$target \mapsto here,$
$fid \mapsto root,$
$type \mapsto$ "backupGetAdopter",
$aid \mapsto actId,$
$finishEnd \mapsto$ FALSE,
$actionType \mapsto$ "live"])

$\land\ waitForMsgs' = waitForMsgs \setminus \{masterWFM\}$

we don't expect backup to die
so we don't add
backupGetAdopterDone in *waitForMsgs*

$\land\ C!IncrMSEQ(1)$
$\land\ pendingAct' = pendingAct$

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

$MasterCompletedDone \triangleq$
$\land\ pstate =$ "running"
$\land\ msgs \neq \{\}$
$\land$ LET $msg \triangleq C!FindMSG($"masterCompletedDone"$)$
IN $\land\ msg \neq C!NotMessage$
$\land$ LET $here \triangleq msg.dst$
$root \triangleq msg.fid$
$success \triangleq msg.success$
$rootPlace \triangleq C!GetFinishHome(root)$
$isAdopter \triangleq msg.isAdopter$

$backupPlace \triangleq msg.backupPlace$
$finishEnd \triangleq msg.finishEnd$
$masterWFM \triangleq [\ src \mapsto rootPlace,$
$dst \mapsto here,$
$target \mapsto here,$
$fid \mapsto root,$
$isAdopter \mapsto isAdopter,$
$type \mapsto$ "masterCompletedDone" $]$
$backupWFM \triangleq [\ src \mapsto backupPlace,$
$dst \mapsto here,$
$fid \mapsto root,$
$target \mapsto here,$
$isAdopter \mapsto isAdopter,$
$type \mapsto$ "backupCompletedDone" $]$

IN $\land\ SetActionNameAndDepth(\langle$"MasterCompletedDone", $here\rangle)$
$\land$ IF $success \land rootPlace \notin killed$
THEN $\land\ C!ReplaceMsg(msg, [mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto backupPlace,$
$target \mapsto here,$
$fid \mapsto root,$
$type \mapsto$ "backupCompleted",
$finishEnd \mapsto finishEnd,$
$isAdopter \mapsto isAdopter])$
$\land$ IF $finishEnd$ THEN $waitForMsgs' = (waitForMsgs \setminus \{masterWFM\})$
ELSE $waitForMsgs' = (waitForMsgs \setminus \{masterWFM\}) \cup \{backupWFM\}$
$\land\ C!IncrMSEQ(1)$
ELSE $\land\ C!ReplaceMsg(msg, [\ mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto C!GetBackup(rootPlace),$
$source \mapsto C!NotPlace,$
$target \mapsto here,$
$fid \mapsto root,$
$type \mapsto$ "backupGetAdopter",
$aid \mapsto C!NotActivity.aid,$
$finishEnd \mapsto$ FALSE,
$actionType \mapsto$ "completed"$])$
$\land\ waitForMsgs' = waitForMsgs \setminus \{masterWFM\}$
we don't expect backup to die
so we don't add *backupGetAdopterDone*
in *waitForMsgs*
$\land\ C!IncrMSEQ(1)$

$\land$ UNCHANGED $\langle convertSet, adoptSet, mastersStatus, fstates, pstate,$
$thrds, pendingAct, killed, fmasters, fbackups,$

$blockedThrds,\ runningThrds\rangle$

$FindBlockedThreadGetAdopterDone \triangleq$
LET $tset \triangleq \{t \in blockedThrds :$
$\land\ t.here \notin killed$
$\land\ thrds[t.here][t.tid].status = \text{“blocked”}$
$\land\ thrds[t.here][t.tid].blockingType = \text{“AsyncTransit”}$
$\land\ C!FindIncomingMSG(t.here,\ \text{“backupGetAdopterDone”}) \neq C!NotMessage\ \}$
IN IF $tset = \{\}$ THEN $C!NotPlaceThread$
ELSE CHOOSE $x \in tset$: TRUE

$GetAdopterDone \triangleq$
$\land\ pstate = \text{“running”}$
$\land\ msgs \neq \{\}$
$\land$ LET $msg \triangleq C!FindMSG(\text{“backupGetAdopterDone”})$
IN $\land\ msg \neq C!NotMessage$
$\land$ LET $here \triangleq msg.dst$
$actionType \triangleq msg.actionType$
$adoptedRoot \triangleq msg.adoptedRoot$
$adoptedRootPlace \triangleq C!GetFinishHome(msg.adoptedRoot)$
$adoptedFID \triangleq msg.fid$
IN $\land\ SetActionNameAndDepth(\langle \text{“GetAdopterDone”},\ here\rangle)$
$\land$ IF $actionType = \text{“transit”}$
THEN $\land\ C!ReplaceMsg(msg, [\ mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto adoptedRootPlace,$
$target \mapsto msg.target,$
$fid \mapsto adoptedRoot,$
$type \mapsto \text{“adopterTransit”},$
$adoptedFID \mapsto adoptedFID])$
$\land\ C!IncrMSEQ(1)$
ELSE IF $actionType = \text{“live”}$
THEN $\land\ C!ReplaceMsg(msg, [mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto adoptedRootPlace,$
$source \mapsto msg.source,$
$target \mapsto msg.target,$
$fid \mapsto adoptedRoot,$
$aid \mapsto msg.aid,$
$type \mapsto \text{“adopterLive”},$
$adoptedFID \mapsto adoptedFID])$
$\land\ C!IncrMSEQ(1)$
ELSE IF $actionType = \text{“completed”}$
THEN $\land\ C!ReplaceMsg(msg, [mid \mapsto seq.mseq,$

```
                                                    src      ↦ here,
                                                    dst      ↦ adoptedRootPlace,
                                                    target   ↦ msg.target,
                                                    fid      ↦ adoptedRoot,
                                                    finishEnd ↦ msg.finishEnd,
                                                    type     ↦ "adopterCompleted",
                                                    adoptedFID ↦ adoptedFID])
                                  ∧ C!IncrMSEQ(1)
                              ELSE  FALSE
  ∧ UNCHANGED ⟨fstates, pstate, thrds, killed, pendingAct,  fmasters, fbackups, waitForMsgs,
        mastersStatus, adoptSet, convertSet, blockedThrds, runningThrds⟩

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

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

 Terminated finish unblocks its thread
UnblockTerminateAsync ≜
  ∧ pstate = "running"
  ∧ LET pthrd ≜ FindBlockedThreadAsyncTerm
    IN  ∧ pthrd ≠ C!NotPlaceThread
        ∧ LET here ≜ pthrd.here
               tid ≜ pthrd.tid
               msg ≜ C!FindIncomingMSG(here, "backupCompletedDone")
               success ≜ msg.success
               top ≜ Head(thrds[here][tid].stack)
               blk ≜ top.b
               fid ≜ top.fid
               root ≜ msg.fid
               rootPlace ≜ 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
  ∧ UNCHANGED ⟨convertSet, adoptSet, mastersStatus, fstates, seq, msgs,
               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⟩)
             ∧ msg ≠ C!NotMessage
             ∧ 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⟩)
```

$\land\ backupPlace \neq C!NotPlace$
$\land\ fid \neq C!NotID$
$\land\ fstates[fid].here = here$
$\land\ target = src$
$\land\ mastersStatus[here].status =$ "running"
$\land$ IF $(fmasters[fid].live[target] > 0 \land fmasters[fid].numActive > 0)$
THEN $\land\ fmasters' = [fmasters$ EXCEPT $![fid].live[target] = @ - 1,$
$![fid].numActive = @ - 1,$
$![fid].isReleased =$
$(fmasters[fid].numActive = 1)]$
ELSE $\land\ target \in killed$
$\land\ fmasters' = fmasters$
$\land$ IF $(fmasters'[fid].numActive = 0 \land src \notin killed)$
THEN $\land\ C!ReplaceMsgSet(msg, \{[mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto src,$
$target \mapsto target,$
$fid \mapsto fid,$
$type \mapsto$ "masterCompletedDone",
$success \mapsto$ TRUE,
$isAdopter \mapsto$ FALSE,
$finishEnd \mapsto finishEnd,$
$backupPlace \mapsto backupPlace],$
$[mid \mapsto seq.mseq + 1,$
$src \mapsto here,$
$dst \mapsto here,$
$fid \mapsto fid,$
$type \mapsto$ "releaseFinish"]\})
$\land\ C!IncrMSEQ(2)$
ELSE IF $fmasters'[fid].numActive = 0$
THEN $\land\ C!ReplaceMsg(msg, [mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto here,$
$fid \mapsto fid,$
$type \mapsto$ "releaseFinish"])
$\land\ C!IncrMSEQ(1)$
ELSE IF $src \notin killed$
THEN $\land\ C!ReplaceMsg(msg, [mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto src,$
$target \mapsto target,$
$fid \mapsto fid,$
$type \mapsto$ "masterCompletedDone",
$success \mapsto$ TRUE,
$isAdopter \mapsto$ FALSE,

$finishEnd \mapsto finishEnd,$
$backupPlace \mapsto backupPlace])$
$\land\ C!IncrMSEQ(1)$
ELSE $\land\ C!RecvMsg(msg)$
$\land\ seq' = seq$
$\land$ UNCHANGED $\langle convertSet,\ adoptSet,\ mastersStatus,\ fstates,\ pstate,$
$thrds,\ killed,\ pendingAct,\ \ fbackups,\ waitForMsgs,$
$blockedThrds,\ runningThrds\rangle$

Adopting *Finish* master replica actions

$AdopterTransit \triangleq$
$\land\ pstate =$ "running"
$\land\ msgs \neq \{\}$
$\land$ LET $msg \triangleq C!FindMSG($"adopterTransit"$)$
IN $\land\ msg \neq C!NotMessage$
$\land$ LET $here \triangleq msg.dst$
$fid \triangleq msg.fid$
$src \triangleq msg.src$
$target \triangleq msg.target$
$backupPlace \triangleq C!GetBackup(here)$
$adoptedFID \triangleq msg.adoptedFID$
IN $\land\ SetActionNameAndDepth(\langle$"AdopterTransit"$, here\rangle)$
$\land\ mastersStatus[here].status =$ "running"
$\land\ fid \neq C!NotID$
$\land\ fstates[fid].here = here$
$\land$ LET $submit \triangleq src \notin killed\ \land\ target \notin killed$
IN $\land$ IF $submit$
THEN $\land\ fmasters' = [fmasters$ EXCEPT $![fid].transitAdopted[src][target] = @ + 1,$
$![fid].numActive = @ + 1]$
ELSE $\land\ fmasters' = fmasters$
$\land\ C!ReplaceMsg(msg,\ [\ mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto src,$
$target \mapsto target,$
$fid \mapsto fid,$
$type \mapsto$ "masterTransitDone",
$submit \mapsto submit,$
$success \mapsto$ TRUE,
$backupPlace \mapsto backupPlace,$
$isAdopter \mapsto$ TRUE,
$adoptedFID \mapsto adoptedFID\ ])$
$\land\ C!IncrMSEQ(1)$
$\land$ UNCHANGED $\langle convertSet,\ adoptSet,\ mastersStatus,\ fstates,\ pstate,$
$thrds,\ killed,\ pendingAct,\ fbackups,\ waitForMsgs,$

```
                    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"
```

$\land\ msgs \neq \{\}$
$\land$ LET $msg \triangleq C!FindMSG(\text{“adopterCompleted”})$
IN $\land\ msg \neq C!NotMessage$
$\land$ LET $here \triangleq msg.dst$
$mid \triangleq msg.mid$
$fid \triangleq msg.fid$
$src \triangleq msg.src$
$target \triangleq msg.target$
$backupPlace \triangleq C!GetBackup(here)$
$finishEnd \triangleq msg.finishEnd$
IN $\land\ SetActionNameAndDepth(\langle \text{“AdopterCompleted”}, here \rangle)$
$\land\ mastersStatus[here].status = \text{“running”}$
$\land\ backupPlace \neq C!NotPlace$
$\land\ fid \neq C!NotID$
$\land\ fstates[fid].here = here$
$\land\ fmasters[fid].liveAdopted[target] > 0$
$\land\ fmasters[fid].numActive > 0$
$\land\ fmasters' = [fmasters$ EXCEPT $![fid].liveAdopted[target] = @ - 1,$
$![fid].numActive = @ - 1,$
$![fid].isReleased = (fmasters[fid].numActive = 1)]$
$\land$ IF $fmasters'[fid].numActive = 0$
THEN $\land\ C!ReplaceMsgSet(msg, \{[mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto src,$
$target \mapsto target,$
$fid \mapsto fid,$
$type \mapsto \text{“masterCompletedDone”},$
$success \mapsto$ TRUE,
$isAdopter \mapsto$ TRUE,
$finishEnd \mapsto finishEnd,$
$backupPlace \mapsto backupPlace],$
$[mid \mapsto seq.mseq + 1,$
$src \mapsto here,$
$dst \mapsto here,$
$fid \mapsto fid,$
$type \mapsto \text{“releaseFinish”}]\})$
$\land\ C!IncrMSEQ(2)$
ELSE IF $finishEnd$
THEN $\land\ C!RecvMsg(msg)$
$\land\ seq' = seq$
ELSE $\land\ C!ReplaceMsg(msg, [mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto src,$
$target \mapsto target,$
$fid \mapsto fid,$

$$
\begin{array}{rl}
 & type \mapsto \text{“masterCompletedDone”}, \\
 & success \mapsto \text{TRUE}, \\
 & isAdopter \mapsto \text{TRUE}, \\
 & finishEnd \mapsto finishEnd, \\
 & backupPlace \mapsto backupPlace]) \\
\wedge\ C!IncrMSEQ(1) & \\
\wedge\ \text{UNCHANGED}\ \langle convertSet,\ adoptSet,\ mastersStatus,\ fstates,\ pstate, & \\
\quad thrds,\ waitForMsgs,\ killed,\ pendingAct,\ fbackups,\ waitForMsgs, & \\
\quad blockedThrds,\ runningThrds\rangle &
\end{array}
$$

Finish backup replica actions

$$
\begin{array}{l}
BackupGetAdopter\ \triangleq \\
\quad \wedge\ pstate = \text{“running”} \\
\quad \wedge\ msgs \neq \{\} \\
\quad \wedge\ \text{LET}\ msg \triangleq C!FindMSG(\text{“backupGetAdopter”}) \\
\quad\quad \text{IN} \quad \wedge\ msg \neq C!NotMessage \\
\quad\quad\quad\quad \wedge\ \text{LET}\ here \triangleq msg.dst \\
\quad\quad\quad\quad\quad\quad fid \triangleq msg.fid \\
\quad\quad\quad\quad\quad\quad src \triangleq msg.src \\
\quad\quad\quad\quad\quad\quad actionType \triangleq msg.actionType \\
\quad\quad\quad\quad\quad\quad source \triangleq msg.source \\
\quad\quad\quad\quad\quad\quad target \triangleq msg.target \\
\quad\quad\quad\quad\quad\quad \text{IN} \quad \wedge\ SetActionNameAndDepth(\langle \text{“BackupGetAdopter”}, here\rangle) \\
\quad\quad\quad\quad\quad\quad\quad\quad \wedge\ fbackups[fid].isAdopted = \text{TRUE} \\
\quad\quad\quad\quad\quad\quad\quad\quad \wedge\ \text{IF}\ src \in killed \vee msg.dst \in killed \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \text{THEN} \quad \wedge\ C!RecvMsg(msg) \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \wedge\ seq' = seq \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \text{ELSE} \quad \wedge\ C!ReplaceMsg(msg, [\ mid \mapsto seq.mseq, \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad src \mapsto here, \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad dst \mapsto src, \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad source \mapsto source, \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad target \mapsto target, \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad fid \mapsto fid, \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad adoptedRoot \mapsto fbackups[fid].adoptedRoot, \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad actionType \mapsto actionType, \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad aid \mapsto msg.aid, \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad finishEnd \mapsto msg.finishEnd, \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad type \mapsto \text{“backupGetAdopterDone”}]) \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \wedge\ C!IncrMSEQ(1) \\
\quad \wedge\ \text{UNCHANGED}\ \langle fstates,\ pstate,\ thrds,\ killed,\ pendingAct,\ fmasters, \\
\quad\quad\quad\quad fbackups,\ waitForMsgs,\ mastersStatus,\ adoptSet,\ convertSet, \\
\quad\quad\quad\quad blockedThrds,\ runningThrds\rangle \\
\\
BackupTransit\ \triangleq
\end{array}
$$

$\land\ pstate = \text{“running”}$
$\land\ msgs \neq \{\}$
$\land$ LET $msg \triangleq C!FindMSG(\text{“backupTransit”})$
IN $\land\ msg \neq C!NotMessage$
$\land$ LET $here \triangleq msg.dst$
$fid \triangleq msg.fid$
$src \triangleq msg.src$
$target \triangleq msg.target$
$isAdopter \triangleq msg.isAdopter$
$adoptedFID \triangleq msg.adoptedFID$
IN $\land\ SetActionNameAndDepth(\langle \text{“BackupTransit”}, here \rangle)$
$\land\ fmasters[fid].backupPlace = here$
$\land$ IF $\neg\ isAdopter \land \neg fbackups[fid].isAdopted$
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 $![fid].transitAdopted[src][target] = @ + 1,$
$![fid].numActive = @ + 1\ ]$
$\land$ IF $fbackups[fid].isAdopted$ Change to the path of *adopterTransit*
THEN $\land\ C!ReplaceMsg(msg, [\ mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto src,$
$target \mapsto target,$
$fid \mapsto fid,$
$type \mapsto \text{“masterTransitDone”},$
$isAdopter \mapsto \text{FALSE},$
$adoptedFID \mapsto C!NotID,$
$backupPlace \mapsto C!NotPlace,$
$submit \mapsto \text{FALSE},$
$success \mapsto \text{FALSE}]\ )$
$\land\ C!IncrMSEQ(1)$
ELSE IF $src \in killed$
THEN $\land\ C!RecvMsg(msg)$
$\land\ seq' = seq$
ELSE $\land\ C!ReplaceMsg(msg, [\ mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto src,$
$target \mapsto target,$
$fid \mapsto fid,$

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

BackupLive ≜
  ∧ pstate = "running"
  ∧ msgs ≠ {}
  ∧ LET msg ≜ C!FindMSG("backupLive")
    IN  ∧ msg ≠ C!NotMessage
        ∧ LET here ≜ msg.dst
              fid ≜ msg.fid
              src ≜ msg.src
              source ≜ msg.source
              target ≜ msg.target
              isAdopter ≜ msg.isAdopter
              adoptedFID ≜ msg.adoptedFID
          IN  ∧ SetActionNameAndDepth(⟨"BackupLive", here⟩)
              ∧ fmasters[fid].backupPlace = here
              ∧ IF ¬ isAdopter ∧ ¬fbackups[fid].isAdopted
                THEN ∧ fbackups[fid].transit[source][target] > 0
                     ∧ fbackups' = [fbackups EXCEPT ![fid].transit[source][target] = @ − 1,
                                                    ![fid].live[target] = @ + 1]
                ELSE ∧ fbackups' = fbackups
              We don't have transitAdopted at the backups!!!!
              ∧ fbackups[fid].transitAdopted[source][target] > 0
              ∧ fbackups' = [ fbackups EXCEPT ![fid].transitAdopted[source][target] = @ −
                                              1, ![fid].liveAdopted[target] = @ + 1 ]
           ∧ IF fbackups[fid].isAdopted  Change to the path of adopterLive
             THEN ∧ C!ReplaceMsg(msg, [ mid  ↦ seq.mseq,
                                        src  ↦ here,
                                        dst  ↦ src,
                                     source  ↦ source,
                                     target  ↦ target,
                                        fid  ↦ fid,
                                        aid  ↦ msg.aid,
                                       type  ↦ "masterLiveDone",
                                     submit  ↦ FALSE,
                                    success  ↦ FALSE,
                                  isAdopter ↦ FALSE,
                                 adoptedFID ↦ C!NotID,
```

```
                                                backupPlace ↦ C!NotPlace])
                    ∧ C!IncrMSEQ(1)
            ELSE IF src ∈ killed
            THEN ∧ C!RecvMsg(msg)
                 ∧ seq′ = seq
            ELSE ∧ C!ReplaceMsg(msg, [ mid      ↦ seq.mseq,
                                       src      ↦ here,
                                       dst      ↦ src,
                                       target   ↦ target,
                                       source   ↦ source,
                                       fid      ↦ fid,
                                       aid      ↦ msg.aid,
                                       type     ↦ "backupLiveDone",
                                       success  ↦ TRUE,
                                       isAdopter ↦ isAdopter,
                                       adoptedFID ↦ adoptedFID])
                    ∧ C!IncrMSEQ(1)
  ∧ UNCHANGED ⟨convertSet, fstates, pstate, thrds, pendingAct, fmasters, waitForMsgs,
               blockedThrds, runningThrds, killed, adoptSet, mastersStatus⟩

BackupCompleted ≜
  ∧ pstate = "running"
  ∧ msgs ≠ {}
  ∧ LET msg ≜ C!FindMSG("backupCompleted")
    IN  ∧ msg ≠ C!NotMessage
        ∧ LET here ≜ msg.dst
              fid ≜ msg.fid
              src ≜ msg.src
              target ≜ msg.target
              isAdopter ≜ msg.isAdopter
              finishEnd ≜ msg.finishEnd
          IN  ∧ fmasters[fid].backupPlace = here
              ∧ SetActionNameAndDepth(⟨"BackupCompleted", here⟩)
              ∧ IF ¬ isAdopter ∧ ¬fbackups[fid].isAdopted
                THEN ∧ fbackups[fid].live[target] > 0
                     ∧ fbackups[fid].numActive > 0
                     ∧ fbackups′ = [fbackups EXCEPT ![fid].live[target] = @ − 1,
                                                    ![fid].numActive = @ − 1]
                ELSE ∧ fbackups′ = fbackups
              ∧ 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,
                                             ![child].adoptedRoot = adopter]
              ∧ fmasters' = [fmasters EXCEPT ![adopter].children = fmasters[adopter].children \ {child},
                                             ![adopter].liveAdopted =
                                                [p ∈ PLACE ↦ fmasters[adopter].liveAdopted[p]
                                                             + fbackups[child].live[p]],
                                             ![adopter].transitAdopted =
```

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad [p \in PLACE \mapsto \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad [q \in PLACE \mapsto fmasters[adopter].transitAdopted[p][q] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad + fbackups[child].transit[p][q]]], \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad ![adopter].numActive = @ + fbackups[child].numActive] \\
\qquad\qquad \wedge\ adoptSet' = adoptSet \setminus \{pair\} \\
\qquad\qquad \wedge\ convertSet' = \ convertSet \cup \{t \in C!ConvTask : \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge\ t.pl \neq C!NotPlace \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge\ t.pl \notin killed \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge\ t.fid = adopter \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge\ t.here = here\} \\
\qquad\qquad \wedge \text{ IF } \exists\, m \in adoptSet' : m.here = here \\
\qquad\qquad\qquad \text{THEN } \wedge\ mastersStatus' = mastersStatus \\
\qquad\qquad\qquad \text{ELSE } \wedge\ mastersStatus' = [mastersStatus \text{ EXCEPT } ![here].status = \text{“convertDead”}] \\
\quad \wedge \text{ UNCHANGED } \langle fstates,\ msgs,\ pstate,\ seq,\ thrds,\ killed,\ pendingAct,\ \ waitForMsgs, \\
\qquad\qquad\qquad blockedThrds,\ runningThrds\rangle
\end{array}
$$

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

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

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad !\,[\mathit{fid}].\mathit{transit}[\mathit{dead}][\mathit{pl}] \;= 0, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad !\,[\mathit{fid}].\mathit{transitAdopted}[\mathit{dead}][\mathit{pl}] \;= 0, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad !\,[\mathit{fid}].\mathit{live}[\mathit{dead}] = 0, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad !\,[\mathit{fid}].\mathit{liveAdopted}[\mathit{dead}] = 0] \\
\qquad\qquad\qquad\qquad \wedge \text{IF } \mathit{fmasters}'[\mathit{fid}].\mathit{numActive} = 0 \\
\qquad\qquad\qquad\qquad\quad \text{THEN } \wedge C\,!\,\mathit{SendMsg}([\mathit{mid} \mapsto \mathit{seq}.\mathit{mseq}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathit{src} \mapsto \mathit{here}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathit{dst} \mapsto \mathit{here}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathit{fid} \mapsto \mathit{fid}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathit{type} \mapsto \text{“releaseFinish”}]) \\
\qquad\qquad\qquad\qquad\qquad\qquad \wedge C\,!\,\mathit{IncrMSEQ}(1) \\
\qquad\qquad\qquad\qquad\quad \text{ELSE } \wedge \mathit{msgs}' = \mathit{msgs} \\
\qquad\qquad\qquad\qquad\qquad\qquad \wedge \mathit{seq}' = \mathit{seq} \\
\qquad\qquad\qquad\qquad \wedge \text{IF } \exists\, m \in \mathit{convertSet}' : m.\mathit{here} = \mathit{here} \\
\qquad\qquad\qquad\qquad\quad \text{THEN } \mathit{mastersStatus}' = \mathit{mastersStatus} \\
\qquad\qquad\qquad\qquad\quad \text{ELSE } \mathit{mastersStatus}' = [\mathit{mastersStatus} \text{ EXCEPT } ![\mathit{here}].\mathit{status} = \text{“running”}] \\
\qquad\qquad \wedge \text{UNCHANGED } \langle \mathit{fstates}, \mathit{pstate}, \mathit{thrds}, \mathit{killed}, \mathit{pendingAct}, \;\; \mathit{fbackups}, \mathit{waitForMsgs}, \\
\qquad\qquad\qquad\qquad\quad \mathit{adoptSet}, \;\; \mathit{blockedThrds}, \mathit{runningThrds}\rangle
\end{array}
$$

$$
\begin{array}{l}
\mathit{FindWaitForMSG} \;\triangleq \\
\quad \text{LET } \mathit{mset} \;\triangleq\; \{m \in \mathit{waitForMsgs} : \\
\qquad\qquad\qquad\qquad \wedge m.\mathit{src} \;\in \mathit{killed} \\
\qquad\qquad\qquad\qquad \wedge m.\mathit{dst} \;\notin \mathit{killed} \\
\qquad\qquad\qquad\qquad \wedge m.\mathit{src} \in \mathit{killed}\} \\
\quad \text{IN} \quad \text{IF } \mathit{mset} = \{\} \text{ THEN } C\,!\,\mathit{NotMessage} \\
\qquad\quad \text{ELSE CHOOSE } x \in \mathit{mset} : \text{TRUE}
\end{array}
$$

$$
\begin{array}{l}
\mathit{SimulateFailedResponse} \;\triangleq \\
\quad \wedge \mathit{pstate} = \text{“running”} \\
\quad \wedge \mathit{killed} \;\neq \{\} \\
\quad \wedge \mathit{waitForMsgs} \neq \{\} \\
\quad \wedge \text{LET } \mathit{msg} \;\triangleq\; \mathit{FindWaitForMSG} \\
\qquad \text{IN} \quad \wedge \mathit{msg} \neq C\,!\,\mathit{NotMessage} \\
\qquad\qquad\quad \wedge \text{LET } \mathit{dead} \;\triangleq\; \mathit{msg}.\mathit{src} \\
\qquad\qquad\qquad\quad \mathit{here} \;\triangleq\; \mathit{msg}.\mathit{dst} \\
\qquad\qquad\qquad\quad \mathit{delMsgs} \;\triangleq\; \{m \in \mathit{msgs} : m.\mathit{dst} = \mathit{dead}\ \} \\
\qquad\qquad\qquad\quad \mathit{wfm} \;\triangleq\; \{m \in \mathit{waitForMsgs} : m.\mathit{dst} = \mathit{dead}\} \\
\qquad\qquad\quad \text{IN} \quad \wedge \mathit{SetActionNameAndDepth}(\langle \text{“SimulateFailedResponse”}, \mathit{here}\rangle) \\
\qquad\qquad\qquad\quad \wedge \mathit{waitForMsgs}' = (\mathit{waitForMsgs} \setminus \mathit{wfm}) \setminus \{\mathit{msg}\} \\
\qquad\qquad\qquad\quad \wedge C\,!\,\mathit{IncrMSEQ}(1) \\
\qquad\qquad\qquad\quad \wedge \text{IF } \mathit{msg}.\mathit{type} = \text{“masterLiveDone”} \\
\qquad\qquad\qquad\qquad\quad \text{THEN IF } \neg(\exists\, m \in \mathit{msgs} : \;\; \text{message has been sent already} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \wedge m.\mathit{type} = \mathit{msg}.\mathit{type} \wedge m.\mathit{src} = \mathit{msg}.\mathit{src} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \wedge m.\mathit{dst} = \mathit{msg}.\mathit{dst} \wedge m.\mathit{fid} = \mathit{msg}.\mathit{fid}
\end{array}
$$

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

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

$type \mapsto$ "backupTransitDone",
$isAdopter \mapsto msg.isAdopter$,
$adoptedFID \mapsto msg.adoptedFID$,
$success \mapsto$ FALSE]}
ELSE $\land msgs' = (msgs \setminus delMsgs)$
ELSE FALSE
$\land$ UNCHANGED $\langle convertSet, adoptSet, mastersStatus, fstates, pstate, thrds, killed, pendingAct, fmasters, fbackups, blockedThrds, runningThrds\rangle$

Predicate enumerating all possible next actions

$Next \triangleq$
$\lor RecvAsync$
$\lor ReleaseRootFinish$
$\lor AuthorizeReceivedAsync$
$\lor BackupTransit$
$\lor BackupLive$
$\lor BackupCompleted$
$\lor BackupGetAdopter$
$\lor MasterTransit$
$\lor MasterLive$
$\lor MasterCompleted$
$\lor MasterTransitDone$
$\lor MasterLiveDone$
$\lor MasterCompletedDone$
$\lor AdopterTransit$
$\lor AdopterLive$
$\lor AdopterCompleted$
$\lor SeekAdoption$
$\lor ConvertDeadActivities$
$\lor SimulateFailedResponse$
$\lor GetAdopterDone$
$\lor RunExprOrKill$
$\lor ScheduleNestedFinish$
$\lor TerminateAsync$
$\lor SpawnRemoteAsync$
$\lor SpawnLocalAsync$
$\lor StopFinish$
$\lor StartFinish$
$\lor AuthorizeTransitAsync$
$\lor UnblockTerminateAsync$

Asserting fairness properties to all actions

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

Specification

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

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

$-$ *metadir* /media/u5482878/$DATAPART1$/*tla_ws*/states -checkpoint 0
$-$ *metadir* /Users/shamouda/*tla_states* $-$ checkpoint 0
**

* Modification History
* Last modified *Mon Dec* 11 20:55:15 *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 u5482878