MODULE *ExecutorDistFinishRep*

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. It implements a buggy replication protocol that was used in $X10$ release 1.4.2, and was used for proof of concept evaluation in $PPoPP14$

$PPoPP14$ wrong replication:

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

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 $0s$
$actionName$,	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

$Finish(fid) \triangleq$ INSTANCE $DistFinish$

$C \triangleq$ INSTANCE $Commons$

$GetRootFinishId(fid) \triangleq$
IF $fid = C!NoParent$ THEN $C!NotID$
ELSE IF $Finish(fid)!IsRoot$ THEN fid
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

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

Initialization

$Init \triangleq$
$\land\ actionName = \langle \text{“Init”},\ PROG\_HOME \rangle$
$\land\ depth = 0$
$\land\ fstates = [r \in\ C!IDRange \mapsto$
$[id \mapsto\ C!NotID,\ status \mapsto \text{“unused”},\ type \mapsto\ \text{“NA”},$
$count \mapsto 0,\ here \mapsto\ C!NotPlace,$
$parent \mapsto\ C!NotID,\ root \mapsto\ C!NotID,\ isGlobal \mapsto$ 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,
$adoptedRoot \mapsto C!NotID,$
$numActive \mapsto 0]]$
$\land\ pstate\ = \text{“running”}$
$\land\ mastersStatus = [p \in PLACE \mapsto [\ status\ \mapsto \text{“running”},$
$lastKilled \mapsto C!NotPlace]]$
$\land\ msgs\ = \{\}$
$\land\ seq\ = [aseq \mapsto 1,\ fseq \mapsto C!FIRST\_ID,\ mseq \mapsto 1]$
$\land\ thrds = [p \in PLACE \mapsto$ start with one running thread at $PROG\_HOME$
$[t \in\ C!ThreadID \mapsto$
IF $p = PROG\_HOME \land t = 0$
THEN $[tid \mapsto t,\ status \mapsto \text{“running”},$
$blockingType \mapsto \text{“NA”},$
$stack \mapsto \langle[\ b\ \mapsto 0,$
$i\ \mapsto$ IF $PROG[0].type = \text{“finish”}$

```
                                                THEN C!I_PRE_FIN_ALLOC
                                                ELSE C!I_START,
                                           fid ↦ C!NoParent]⟩]
                    ELSE [tid ↦ t, status ↦ "idle",
                          blockingType ↦ "NA",
                          stack ↦ ⟨⟩]]]
  ∧ runningThrds = {[here ↦ PROG_HOME, tid ↦ 0]}
  ∧ blockedThrds  = {}
  ∧ killed = {}
  ∧ pendingAct = {}
  ∧ waitForMsgs = {}
  ∧ adoptSet = {}
  ∧ convertSet = {}
```

Helper Actions

```
SetActionNameAndDepth(name) ≜
  IF depth = DEPTH THEN TRUE ELSE  ∧ actionName' = name ∧ depth' = depth + 1

FindPendingActivity(actId) ≜
  LET aset ≜ {a ∈ pendingAct : a.aid  = actId}
  IN  IF aset = {} THEN C!NotActivity
      ELSE CHOOSE x ∈ aset : TRUE

FindIdleThread(here) ≜
  LET idleThreads ≜ C!PlaceThread \ (runningThrds ∪ blockedThrds)
      tset ≜ {t ∈ idleThreads :
                ∧ t.here = here
                ∧ t.here ∉ killed
                ∧ thrds[t.here][t.tid].status = "idle"}
  IN  IF tset = {} THEN C!NotPlaceThread
      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}]
                     ∧ 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"}
```

$\land\ curStmt \geq 0$
$\land\ curStmt \leq PROG[blk].mxstmt$
$\land\ PROG[nested].type = \text{“finish”}$
$\land\ PROG[nested].dst = t.here\ \}$

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 $\land\ SetActionNameAndDepth(\langle \text{“ScheduleNestedFinish”}, here\rangle)$
$\land\ thrds' = [thrds$ EXCEPT $![here][tid].stack =$
$\langle[\ b \mapsto nested,$
$i \mapsto C!I\_START,$
$fid \mapsto newFid],$
$[\ b \mapsto top.b,$
$i \mapsto curStmt,$
$fid \mapsto fid]$
$\rangle \circ tail]$
$\land\ Finish(seq.fseq)!Alloc(C!ROOT\_FINISH, here, fid, newFid)$
$\land\ C!IncrFSEQ$
$\land\ fmasters' = [fmasters$ EXCEPT $![encRoot].children =$
$@ \cup \{newFid\}]$
$\land\ fbackups' = [fbackups$ EXCEPT $![encRoot].children =$
$@ \cup \{newFid\}]$
$\land$ UNCHANGED $\langle convertSet, adoptSet, mastersStatus, msgs, pstate, waitForMsgs,$
$killed, pendingAct, runningThrds, blockedThrds\rangle$

$FindRunningThreadForSpawnLocalAsync \triangleq$
LET $tset \triangleq \{t \in runningThrds :$
$\land\ t.here \notin killed$

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

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

$SpawnLocalAsync \triangleq$
$\wedge\ pstate = \text{“running”}$
$\wedge$ LET $pthrd \triangleq FindRunningThreadForSpawnLocalAsync$
IN $\wedge\ pthrd \neq C!NotPlaceThread$
$\wedge$ 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 $\wedge\ SetActionNameAndDepth(\langle \text{“SpawnLocalAsync”}, here\rangle)$
$\wedge$ IF $act.fid \neq C!NoParent$
THEN $Finish(act.fid)!NotifyLocalActivitySpawnAndCreation(here, act)$
ELSE $fstates' = fstates$
$\wedge\ C!IncrASEQ$
$\wedge\ thrds' = [thrds$ EXCEPT $![here][tid].stack =$
$\langle[\ b \mapsto top.b,$
$i \mapsto curStmt,$
$fid \mapsto fid]$
$\rangle \circ tail,$
$![here][idle.tid].stack = \langle stkEntry\rangle,$
$![here][idle.tid].status = \text{“running”}]$
$\wedge\ runningThrds' = runningThrds \cup \{[here \mapsto here,\ tid \mapsto idle.tid]\}$
$\wedge$ UNCHANGED $\langle convertSet, adoptSet, mastersStatus, msgs, pstate, killed,$

$pendingAct$, $fmasters$, $fbackups$, $waitForMsgs$, $blockedThrds\rangle$

$FindRunningThreadForSpawnRemoteAsync \triangleq$
LET $tset \triangleq \{t \in runningThrds :$
$\land\ t.here \notin killed$
$\land\ thrds[t.here][t.tid].status =$ "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 $\land\ PROG[blk].type \notin \{$"expr", "kill"$\}$
$\land\ fid \neq C!NoParent$
$\land\ curStmt \geq 0$
$\land\ curStmt \leq PROG[blk].mxstmt$
$\land\ PROG[nested].type =$ "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
$SpawnRemoteAsync \triangleq$
$\land\ pstate =$ "running"
$\land$ LET $pthrd \triangleq FindRunningThreadForSpawnRemoteAsync$
IN $\land\ pthrd \neq C!NotPlaceThread$
$\land$ LET $here \triangleq pthrd.here$
$tid \triangleq pthrd.tid$
$top \triangleq Head(thrds[here][tid].stack)$
$tail \triangleq Tail(thrds[here][tid].stack)$
$lstStmt \triangleq top.i$
$curStmt \triangleq top.i + 1$
$blk \triangleq top.b$
$fid \triangleq top.fid$
$root \triangleq GetRootFinishId(fid)$
$nested \triangleq PROG[blk].stmts[curStmt]$
$dst \triangleq PROG[nested].dst$
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 $\langle convertSet, adoptSet, mastersStatus, pstate, killed, pendingAct,$

$fmasters, fbackups\rangle$

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

$$
\begin{array}{l}
Kill(dead) \triangleq \\
\quad \land\ killed' = killed \cup \{dead\} \\
\quad \land\ adoptSet' = adoptSet \cup \{m \in C!Adopter : \\
\qquad\qquad\qquad\qquad \land\ m.child \neq C!NotID \\
\qquad\qquad\qquad\qquad \land\ m.adopter \neq C!NotID \\
\qquad\qquad\qquad\qquad \land\ m.here \neq dead \\
\qquad\qquad\qquad\qquad \land\ m.here = fstates[m.adopter].here \\
\qquad\qquad\qquad\qquad \land\ m.child \in fmasters[m.adopter].children \\
\qquad\qquad\qquad\qquad \land\ fbackups[m.child].isAdopted = \text{FALSE} \\
\qquad\qquad\qquad\qquad \land\ fstates[m.child].here = dead \\
\qquad\qquad\qquad\qquad \land\ m.adopter = fstates[m.child].eroot\} \\
\quad \land \text{ IF } adoptSet' = \{\} \\
\qquad \text{THEN} \quad \land\ mastersStatus' = [mastersStatus \text{ EXCEPT } ![PROG\_HOME].status = \text{“convertDead”}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad ![PROG\_HOME].lastKilled = dead] \\
\qquad \text{ELSE} \quad \land\ mastersStatus' = [p \in PLACE \mapsto \text{IF } \exists\, m \in adoptSet' : m.here = p \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{THEN } [\ \ status \mapsto \text{“seekAdoption”}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ lastKilled \mapsto dead] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{ELSE } [\ \ status \mapsto \text{“running”}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ lastKilled \mapsto C!NotPlace]] \\
\quad \land\ convertSet' = \{t \in C!ConvTask : \\
\qquad\qquad\qquad \land\ t.pl \neq C!NotPlace \\
\qquad\qquad\qquad \land\ t.pl \neq dead \\
\qquad\qquad\qquad \land\ t.pl \notin killed \\
\qquad\qquad\qquad \land\ t.fid \in \{id \in C!IDRange : \\
\qquad\qquad\qquad\qquad\qquad\quad \land\ fmasters[id].id \neq C!NotID \\
\qquad\qquad\qquad\qquad\qquad\quad \land\ fstates[id].here \neq dead\} \\
\qquad\qquad\qquad \land\ t.here = fstates[t.fid].here\}
\end{array}
$$

```
  ∧ 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"
```

$\land\ PROG[blk].mxstmt = top.i\ \}$

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", $here\rangle)$
$\land\ Finish(fid)!NotifyActivityTermination(\text{FALSE})$
$\land\ thrds' = [thrds$ 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 $x \in tset$: TRUE

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

$StopFinish \triangleq$
$\land\ pstate =$ "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 $\land\ SetActionNameAndDepth(\langle$"StopFinish", $here\rangle)$
$\land\ PROG[top.b].type =$ "finish"
$\land\ PROG[top.b].mxstmt = top.i$
$\land\ Finish(top.fid)!NotifyActivityTermination(\text{TRUE})$

```
                    ∧ 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⟩

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

MasterTransitDone ≜
  ∧ pstate = "running"
  ∧ msgs ≠ {}
  ∧ LET msg ≜ C!FindMSG("backupTransitDone")
    IN  ∧ msg ≠ C!NotMessage
        ∧ msg.dst ∉ killed
        ∧ LET here ≜ msg.dst
              mid ≜ msg.mid
              fid ≜ msg.fid
              source ≜ msg.source
              target ≜ msg.target
              src ≜ msg.src
          IN  ∧ SetActionNameAndDepth(⟨"MasterTransitDone", here⟩)
```

$\land$ $mastersStatus[here].status$ = "running"
$\land$ $fstates[fid].here = here$
$\land$ $waitForMsgs' = waitForMsgs \setminus \{[src \mapsto src,$
$dst \mapsto here,$
$fid \mapsto fid,$
$source \mapsto source,$
$target \mapsto target,$
$type \mapsto$ "backupTransitDone" $]\}$
$\land$ IF $source \in killed$
THEN $\land$ $C!RecvMsg(msg)$
$\land$ $seq' = seq$
ELSE $\land$ $C!ReplaceMsg(msg,$
$[mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto source,$
$target \mapsto target,$
$fid \mapsto fid,$
$type \mapsto$ "masterTransitDone",
$isAdopted \mapsto$ FALSE,
$adoptedRoot \mapsto C!NotID,$
$adoptedFID \mapsto C!NotID,$
$success \mapsto$ TRUE$])$
$\land$ $C!IncrMSEQ(1)$
$\land$ UNCHANGED $\langle convertSet, adoptSet, mastersStatus, fstates, pstate,$
$thrds, killed, pendingAct, fmasters, fbackups,$
$blockedThrds, runningThrds\rangle$

$MasterLiveDone \triangleq$
$\land$ $pstate$ = "running"
$\land$ $msgs \neq \{\}$
$\land$ LET $msg \triangleq C!FindMSG($"backupLiveDone"$)$
IN $\land$ $msg \neq C!NotMessage$
$\land$ $msg.dst \notin killed$
$\land$ LET $here \triangleq msg.dst$
$mid \triangleq msg.mid$
$fid \triangleq msg.fid$
$source \triangleq msg.source$
$target \triangleq msg.target$
$src \triangleq msg.src$
IN $\land$ $SetActionNameAndDepth(\langle$"MasterLiveDone", $here\rangle)$
$\land$ $mastersStatus[here].status$ = "running"
$\land$ $fstates[fid].here = here$
$\land$ IF $target \in killed$
THEN $\land$ $C!RecvMsg(msg)$
$\land$ $seq' = seq$

```
                ELSE  ∧ C!ReplaceMsg(msg,
                                     [ mid    ↦ seq.mseq,
                                       src    ↦ here,
                                       dst    ↦ target,
                                       target ↦ target,
                                       fid    ↦ fid,
                                       aid    ↦ msg.aid,
                                       type   ↦ "masterLiveDone",
                                       isAdopted   ↦ FALSE,
                                       adoptedRoot ↦ C!NotID,
                                       source ↦ source,
                                       submit ↦ TRUE,
                                       success ↦ TRUE])
                      ∧ C!IncrMSEQ(1)
                ∧ waitForMsgs' = waitForMsgs \ {[src    ↦ src,
                                                 dst    ↦ here,
                                                 fid    ↦ fid,
                                                 source ↦ source,
                                                 target ↦ target,
                                                 aid    ↦ msg.aid,
                                                 type   ↦ "backupLiveDone" ]}
  ∧ UNCHANGED ⟨convertSet, adoptSet, mastersStatus,
               fstates, pstate, thrds, killed, pendingAct, fmasters, fbackups,
               blockedThrds, runningThrds⟩

MasterCompletedDone ≜
  ∧ pstate = "running"
  ∧ msgs ≠ {}
  ∧ LET msg ≜ C!FindMSG("backupCompletedDone")
    IN  ∧ msg ≠ C!NotMessage
        ∧ LET here ≜ msg.dst
               mid ≜ msg.mid
               fid ≜ msg.fid
            target ≜ msg.target
               src ≜ msg.src
         finishEnd ≜ msg.finishEnd
          IN  ∧ SetActionNameAndDepth(⟨"MasterCompletedDone", here⟩)
              ∧ mastersStatus[here].status = "running"
              ∧ fstates[fid].here = here
              ∧ waitForMsgs' = waitForMsgs \ {[src    ↦ src,
                                               dst    ↦ here,
                                               fid    ↦ fid,
                                               target ↦ target,
                                               finishEnd ↦ finishEnd,
                                               type   ↦ "backupCompletedDone" ]}
```

$$
\begin{array}{l}
\land \text{ IF } fmasters[fid].isReleased \\
\quad \text{THEN } \land C!RecvMsg(msg) \\
\qquad\quad \land seq' = seq \\
\qquad\quad \land fmasters' = fmasters \\
\quad \text{ELSE } \land \text{IF } fmasters[fid].numActive = 0 \\
\qquad\quad \text{THEN IF } target = here \lor target \in killed \\
\qquad\qquad\quad \text{THEN } \land C!ReplaceMsg(msg, \\
\qquad\qquad\qquad\qquad\qquad [mid \mapsto seq.mseq, \\
\qquad\qquad\qquad\qquad\qquad\ src \mapsto here, \\
\qquad\qquad\qquad\qquad\qquad\ dst \mapsto here, \\
\qquad\qquad\qquad\qquad\qquad\ fid \mapsto fid, \\
\qquad\qquad\qquad\qquad\qquad\ type \mapsto \text{“releaseFinish”}]) \\
\qquad\qquad\qquad\quad \land C!IncrMSEQ(1) \\
\qquad\qquad\qquad\quad \land fmasters' = [fmasters \text{ EXCEPT } ![fid].isReleased = \text{TRUE}] \\
\qquad\qquad\quad \text{ELSE } \land C!ReplaceMsgSet(msg, \{ \\
\qquad\qquad\qquad\qquad\qquad [\ mid \mapsto seq.mseq, \\
\qquad\qquad\qquad\qquad\qquad\ \ src \mapsto here, \\
\qquad\qquad\qquad\qquad\qquad\ \ dst \mapsto target, \\
\qquad\qquad\qquad\qquad\qquad target \mapsto target, \\
\qquad\qquad\qquad\qquad\qquad\ \ fid \mapsto fid, \\
\qquad\qquad\qquad\qquad\qquad\ type \mapsto \text{“masterCompletedDone”}, \\
\qquad\qquad\qquad\qquad finishEnd \mapsto finishEnd, \\
\qquad\qquad\qquad\qquad isAdopted \mapsto \text{FALSE}, \\
\qquad\qquad\qquad\qquad adoptedRoot \mapsto C!NotID, \\
\qquad\qquad\qquad\qquad numActive \mapsto fmasters[fid].numActive, \\
\qquad\qquad\qquad\qquad\ success \mapsto \text{TRUE}], \\
\qquad\qquad\qquad\qquad\quad [mid \mapsto seq.mseq + 1, \\
\qquad\qquad\qquad\qquad\qquad\ src \mapsto here, \\
\qquad\qquad\qquad\qquad\qquad\ dst \mapsto here, \\
\qquad\qquad\qquad\qquad\qquad\ fid \mapsto fid, \\
\qquad\qquad\qquad\qquad\qquad\ type \mapsto \text{“releaseFinish”}]\}) \\
\qquad\qquad\qquad\quad \land C!IncrMSEQ(2) \\
\qquad\qquad\qquad\quad \land fmasters' = [fmasters \text{ EXCEPT } ![fid].isReleased = \text{TRUE}] \\
\qquad\qquad \text{ELSE IF } finishEnd \lor target \in killed \\
\qquad\qquad\quad \text{THEN } \land C!RecvMsg(msg) \\
\qquad\qquad\qquad\quad \land seq' = seq \\
\qquad\qquad\qquad\quad \land fmasters' = fmasters \\
\qquad\qquad\quad \text{ELSE } \land C!ReplaceMsg(msg, \\
\qquad\qquad\qquad\qquad\qquad [\ mid \mapsto seq.mseq, \\
\qquad\qquad\qquad\qquad\qquad\ \ src \mapsto here, \\
\qquad\qquad\qquad\qquad\qquad\ \ dst \mapsto target, \\
\qquad\qquad\qquad\qquad\qquad target \mapsto target, \\
\qquad\qquad\qquad\qquad\qquad\ \ fid \mapsto fid, \\
\qquad\qquad\qquad\qquad numActive \mapsto fmasters[fid].numActive, \\
\qquad\qquad\qquad\qquad isAdopted \mapsto \text{FALSE},
\end{array}
$$

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

$$
\begin{array}{l}
FindBlockedThreadAsyncTerm \triangleq \\
\quad \text{LET}\ tset \triangleq \{t \in blockedThrds : \\
\qquad \wedge\ t.here \notin killed \\
\qquad \wedge\ thrds[t.here][t.tid].status = \text{“blocked”} \\
\qquad \wedge\ thrds[t.here][t.tid].blockingType = \text{“AsyncTerm”} \\
\qquad \wedge\ \text{LET}\ msg \triangleq C!FindIncomingMSG(t.here,\ \text{“masterCompletedDone”}) \\
\qquad\qquad top \triangleq Head(thrds[t.here][t.tid].stack) \\
\qquad\qquad blk \triangleq top.b \\
\qquad\ \text{IN}\ \wedge\ msg \neq C!NotMessage \\
\qquad\qquad \wedge\ PROG[blk].type = \text{“async”} \\
\qquad\qquad \wedge\ PROG[blk].mxstmt = top.i \\
\qquad\qquad \wedge\ msg.fid = fstates[top.fid].root\} \\
\quad \text{IN}\ \ \text{IF}\ tset = \{\}\ \text{THEN}\ C!NotPlaceThread \\
\qquad \text{ELSE}\ \ \text{CHOOSE}\ x \in tset : \text{TRUE}
\end{array}
$$

Terminated finish unblocks its thread

$$
\begin{array}{l}
UnblockTerminateAsync \triangleq \\
\quad \wedge\ pstate = \text{“running”} \\
\quad \wedge\ \text{LET}\ pthrd \triangleq FindBlockedThreadAsyncTerm \\
\qquad \text{IN}\ \ \wedge\ pthrd \neq C!NotPlaceThread \\
\qquad\qquad \wedge\ \text{LET}\ here \triangleq pthrd.here \\
\qquad\qquad\qquad tid \triangleq pthrd.tid \\
\qquad\qquad\qquad msg \triangleq C!FindIncomingMSG(here,\ \text{“masterCompletedDone”}) \\
\qquad\qquad\qquad success \triangleq msg.success \\
\qquad\qquad\qquad top \triangleq Head(thrds[here][tid].stack) \\
\qquad\qquad\qquad blk \triangleq top.b \\
\qquad\qquad\qquad fid \triangleq top.fid \\
\qquad\qquad\qquad root \triangleq msg.fid \\
\qquad\qquad\qquad rootPlace \triangleq C!GetFinishHome(root) \\
\qquad\qquad\qquad finishEnd \triangleq msg.finishEnd \\
\qquad\qquad\ \text{IN}\ \ \wedge\ SetActionNameAndDepth(\langle \text{“UnblockTerminateAsync”},\ here\rangle) \\
\qquad\qquad\qquad \wedge\ \text{IF}\ success \\
\qquad\qquad\qquad\quad \text{THEN}\ \ \wedge\ waitForMsgs' = waitForMsgs \setminus \{[src \mapsto rootPlace, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad dst \mapsto here,
\end{array}
$$

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{target} \mapsto \mathit{here}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{fid} \mapsto \mathit{root}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{finishEnd} \mapsto \mathit{finishEnd}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{type} \mapsto \text{“masterCompletedDone”}\ ]\} \\
\qquad \land \text{ IF } \quad \mathit{Len}(\mathit{thrds}[\mathit{here}][\mathit{tid}].\mathit{stack}) = 1 \\
\qquad\quad \text{THEN} \quad \land \mathit{thrds}' = [\mathit{thrds} \text{ EXCEPT } ![\mathit{here}][\mathit{tid}].\mathit{stack} = \langle\rangle, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad ![\mathit{here}][\mathit{tid}].\mathit{status} = \text{“idle”}] \\
\qquad\qquad\qquad \land \mathit{blockedThrds}' = \mathit{blockedThrds} \setminus \{[\mathit{here} \mapsto \mathit{here},\ \mathit{tid} \mapsto \mathit{tid}]\} \\
\qquad\qquad\qquad \land \mathit{runningThrds}' = \mathit{runningThrds} \\
\qquad\quad \text{ELSE} \quad \land \mathit{thrds}' = [\mathit{thrds} \text{ EXCEPT } ![\mathit{here}][\mathit{tid}].\mathit{stack} = \mathit{Tail}(@), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad ![\mathit{here}][\mathit{tid}].\mathit{status} = \text{“running”}] \\
\qquad\qquad\qquad \land \mathit{blockedThrds}' = \mathit{blockedThrds} \setminus \{[\mathit{here} \mapsto \mathit{here},\ \mathit{tid} \mapsto \mathit{tid}]\} \\
\qquad\qquad\qquad \land \mathit{runningThrds}' = \mathit{runningThrds} \cup \ \{[\mathit{here} \mapsto \mathit{here},\ \mathit{tid} \mapsto \mathit{tid}]\} \\
\qquad \land \text{ IF } \quad \mathit{blk} = 0 \\
\qquad\quad \text{THEN } \mathit{pstate}' = \text{“terminated”} \\
\qquad\quad \text{ELSE } \mathit{pstate}' = \mathit{pstate} \\
\qquad \land \mathit{msgs}' = \mathit{msgs} \\
\qquad \land \mathit{seq}' = \mathit{seq} \\
\text{ELSE IF } \mathit{msg}.\mathit{isAdopted} \\
\text{THEN } \land C!\mathit{ReplaceMsg}(\mathit{msg}, \\
\qquad\qquad\quad [\mathit{mid} \mapsto \mathit{seq}.\mathit{mseq}, \\
\qquad\qquad\quad \mathit{src} \mapsto \mathit{here}, \\
\qquad\qquad\quad \mathit{dst} \mapsto C!\mathit{GetFinishHome}(\mathit{msg}.\mathit{adoptedRoot}), \\
\qquad\qquad\quad \mathit{target} \mapsto \mathit{here}, \\
\qquad\qquad\quad \mathit{fid} \mapsto \mathit{msg}.\mathit{adoptedRoot}, \\
\qquad\qquad\quad \mathit{finishEnd} \mapsto \mathit{finishEnd}, \\
\qquad\qquad\quad \mathit{type} \mapsto \text{“adopterCompleted”}]) \\
\qquad \land \mathit{pstate}' = \mathit{pstate} \\
\qquad \land \mathit{thrds}' = \mathit{thrds} \\
\qquad \land \mathit{waitForMsgs}' = \mathit{waitForMsgs} \\
\qquad \land C!\mathit{IncrMSEQ}(1) \\
\qquad \land \mathit{blockedThrds}' = \mathit{blockedThrds} \\
\qquad \land \mathit{runningThrds}' = \mathit{runningThrds} \\
\text{ELSE } \land C!\mathit{ReplaceMsg}(\mathit{msg}, \\
\qquad\qquad\quad [\mathit{mid} \mapsto \mathit{seq}.\mathit{mseq}, \\
\qquad\qquad\quad \mathit{src} \mapsto \mathit{here}, \\
\qquad\qquad\quad \mathit{dst} \mapsto C!\mathit{GetBackup}(\mathit{rootPlace}), \\
\qquad\qquad\quad \mathit{target} \mapsto \mathit{msg}.\mathit{target}, \\
\qquad\qquad\quad \mathit{fid} \mapsto \mathit{root}, \\
\qquad\qquad\quad \mathit{finishEnd} \mapsto \mathit{finishEnd}, \\
\qquad\qquad\quad \mathit{type} \mapsto \text{“backupCompleted”}]) \\
\qquad \land \mathit{pstate}' = \mathit{pstate} \\
\qquad \land \mathit{thrds}' = \mathit{thrds} \\
\qquad \land \mathit{waitForMsgs}' = \mathit{waitForMsgs} \\
\qquad \land C!\mathit{IncrMSEQ}(1)
\end{array}
$$

$\wedge\ blockedThrds' = blockedThrds$
$\wedge\ runningThrds' = runningThrds$
$\wedge$ UNCHANGED $\langle convertSet,\ adoptSet,\ mastersStatus,$
$fstates,\ killed,\ pendingAct,\ fmasters,\ fbackups\rangle$

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

$AuthorizeTransitAsync \triangleq$
$\wedge\ pstate =$ "running"
$\wedge\ msgs \neq \{\}$
$\wedge$ LET $pthrd \triangleq FindBlockedThreadAuthorizeTransitAsync$
IN $\wedge\ pthrd \neq C!NotPlaceThread$
$\wedge$ LET $here \triangleq pthrd.here$
$tid \triangleq pthrd.tid$
$msg \triangleq C!FindIncomingMSG(here,$ "masterTransitDone")
$success \triangleq msg.success$
$top \triangleq Head(thrds[here][tid].stack)$
$tail \triangleq Tail(thrds[here][tid].stack)$
$lstStmt \triangleq top.i$
$curStmt \triangleq top.i + 1$
$blk \triangleq top.b$
$root \triangleq msg.fid$
$fid \triangleq top.fid$
$rootPlace \triangleq C!GetFinishHome(root)$
$nested \triangleq PROG[blk].stmts[curStmt]$
$asyncDst \triangleq PROG[nested].dst$
$realFID \triangleq$ IF $msg.adoptedFID \neq C!NotID$ THEN $msg.adoptedFID$ ELSE $root$
IN $\wedge\ SetActionNameAndDepth(\langle$"AuthorizeTransitAsync", $here,$ "to",
$asyncDst,$ "success", $success\rangle)$
$\wedge$ IF $success$
THEN $\wedge\ msg.src = rootPlace$
$\wedge\ C!ReplaceMsg(msg,$
$[mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto asyncDst,$
$type \mapsto$ "async",
$fid \mapsto realFID,$

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad b \mapsto nested])\\
\qquad\qquad \wedge\ C!IncrMSEQ(1)\\
\qquad\qquad \wedge\ thrds' = [thrds \text{ EXCEPT } ![here][tid].status = \text{“running”},\\
\qquad\qquad\qquad\qquad\qquad\qquad ![here][tid].stack =\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \langle[\ b \mapsto top.b,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \ i \mapsto curStmt,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \ fid \mapsto fid]\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \rangle \circ tail]\\
\qquad\qquad \wedge\ blockedThrds' = blockedThrds \setminus \{[here \mapsto here,\ tid \mapsto tid]\}\\
\qquad\qquad \wedge\ runningThrds' = runningThrds \cup \{[here \mapsto here,\ tid \mapsto tid]\}\\
\qquad\qquad \wedge\ waitForMsgs' = waitForMsgs \setminus \{[src \mapsto rootPlace,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad dst \mapsto here,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad fid \mapsto root,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad target \mapsto PROG[nested].dst,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad type \mapsto \text{“masterTransitDone”}\ ]\}\\
\text{ELSE IF } msg.isAdopted\\
\text{THEN } \wedge\ C!IncrMSEQ(1)\\
\qquad\qquad \wedge\ thrds' = thrds\\
\qquad\qquad \wedge\ C!ReplaceMsg(msg,\\
\qquad\qquad\qquad\qquad [\ mid \mapsto seq.mseq,\\
\qquad\qquad\qquad\qquad\ \ src \mapsto here,\\
\qquad\qquad\qquad\qquad\ \ dst \mapsto C!GetFinishHome(msg.adoptedRoot),\\
\qquad\qquad\qquad\qquad target \mapsto msg.target,\\
\qquad\qquad\qquad\qquad\ \ fid \mapsto msg.adoptedRoot,\\
\qquad\qquad\qquad\quad adoptedFID \mapsto root,\\
\qquad\qquad\qquad\qquad type \mapsto \text{“adopterTransit”}])\\
\qquad\qquad \wedge\ waitForMsgs' = waitForMsgs\\
\qquad\qquad \wedge\ blockedThrds' = blockedThrds\\
\qquad\qquad \wedge\ runningThrds' = runningThrds\\
\text{ELSE } \wedge\ C!IncrMSEQ(1)\\
\qquad\qquad \wedge\ thrds' = thrds\\
\qquad\qquad \wedge\ C!ReplaceMsg(msg,\\
\qquad\qquad\qquad\qquad [\ mid \mapsto seq.mseq,\\
\qquad\qquad\qquad\qquad\ \ src \mapsto here,\\
\qquad\qquad\qquad\qquad\ \ dst \mapsto C!GetBackup(rootPlace),\\
\qquad\qquad\qquad\qquad source \mapsto here,\\
\qquad\qquad\qquad\qquad target \mapsto msg.target,\\
\qquad\qquad\qquad\qquad\ \ fid \mapsto fid,\\
\qquad\qquad\qquad\qquad type \mapsto \text{“backupTransit”}])\\
\qquad\qquad \wedge\ waitForMsgs' = waitForMsgs\\
\qquad\qquad \wedge\ blockedThrds' = blockedThrds\\
\qquad\qquad \wedge\ runningThrds' = runningThrds\\
\wedge \text{ UNCHANGED } \langle convertSet,\ adoptSet,\ mastersStatus,\ fstates,\ pstate,\\
\qquad\qquad killed,\ pendingAct,\ fmasters,\ fbackups\rangle
\end{array}
$$

```
AuthorizeReceivedAsync ≜
  ∧ pstate = "running"
  ∧ pendingAct ≠ {}
  ∧ msgs ≠ {}
  ∧ LET msg ≜ C!FindMSG("masterLiveDone")
    IN  ∧ msg ≠ C!NotMessage
        ∧ LET here ≜ msg.dst
              actId ≜ msg.aid
              activity ≜ FindPendingActivity(actId)
              root ≜ msg.fid
              submit ≜ msg.submit
              success ≜ msg.success
              rootPlace ≜ C!GetFinishHome(root)
          IN  ∧ SetActionNameAndDepth(⟨"AuthorizeReceivedAsync", here⟩)
              ∧ activity ≠ C!NotActivity
              ∧ fstates[activity.fid].here = here
              ∧ waitForMsgs' = waitForMsgs \ {[src    ↦ rootPlace,
                                               dst    ↦ here,
                                               fid    ↦ root,
                                               aid    ↦ actId,
                                               source ↦ msg.source,
                                               type   ↦ "masterLiveDone" ]}
              ∧ IF success
                THEN  ∧ C!RecvMsg(msg)
                      ∧ pendingAct' = pendingAct \ {activity}
                      ∧ seq' = seq
                      ∧ IF submit
                        THEN LET idleThrd ≜ FindIdleThread(here)
                                 stkEntry ≜ [b ↦ activity.b, i ↦ −1, fid ↦ activity.fid]
                             IN  ∧ thrds' = [thrds EXCEPT ![here][idleThrd.tid].stack = ⟨stkEntry⟩,
                                                          ![here][idleThrd.tid].status = "running"]
                                 ∧ runningThrds' = runningThrds ∪ {[here ↦ here, tid ↦ idleThrd.
                        ELSE  ∧ thrds' = thrds
                              ∧ runningThrds' = runningThrds
                ELSE IF msg.isAdopted
                THEN  ∧ pendingAct' = pendingAct
                      ∧ thrds' = thrds
                      ∧ runningThrds' = runningThrds
                      ∧ C!ReplaceMsg(msg,
                                     [mid    ↦ seq.mseq,
                                      src    ↦ here,
                                      source ↦ msg.source,
                                      target ↦ here,
                                      dst    ↦ C!GetFinishHome(msg.adoptedRoot),
                                      fid    ↦ msg.adoptedRoot,   always refer to the root state
```

```
                                    aid  ↦ actId,
                                    type ↦ "adopterLive"])
                          ∧ C!IncrMSEQ(1)
                ELSE      ∧ pendingAct' = pendingAct
                          ∧ thrds' = thrds
                          ∧ runningThrds' = runningThrds
                          ∧ C!ReplaceMsg(msg,
                                  [ mid  ↦ seq.mseq,
                                    src  ↦ here,
                                    dst  ↦ C!GetBackup(rootPlace),
                                    source ↦ msg.source,
                                    target ↦ msg.target,
                                    fid  ↦ msg.fid,
                                    aid  ↦ actId,
                                    type ↦ "backupLive"])
                          ∧ C!IncrMSEQ(1)
  ∧ UNCHANGED ⟨convertSet, adoptSet, mastersStatus,
               fstates, pstate, 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"]
```

$$
\begin{array}{l}
\qquad\qquad \wedge\ blockedThrds' \ = blockedThrds \setminus \{[here \mapsto here,\ tid \mapsto tid]\} \\
\qquad\qquad \wedge\ runningThrds' = runningThrds \cup \{[here \mapsto here,\ tid \mapsto tid]\} \\
\quad \wedge \text{IF}\ \ blk = 0 \\
\qquad \text{THEN}\ pstate' = \text{“terminated”} \\
\qquad \text{ELSE}\ pstate' = pstate
\end{array}
$$

$$
\begin{array}{l}
ReleaseRootFinish\ \triangleq \\
\quad \wedge\ pstate = \text{“running”} \\
\quad \wedge\ msgs \neq \{\} \\
\quad \wedge\ blockedThrds \neq \{\} \\
\quad \wedge \text{LET}\ msg \triangleq C!FindMSG(\text{“releaseFinish”}) \\
\quad\ \ \text{IN}\ \ \wedge\ msg \neq C!NotMessage \\
\qquad\qquad \wedge \text{LET}\ here \triangleq msg.dst \\
\qquad\qquad\qquad\ \ root \triangleq msg.fid \\
\qquad\qquad\qquad\ \ pthrd \triangleq FindBlockedThreadStopFinish(here,\ root) \\
\qquad\qquad\qquad\ \ tid \triangleq pthrd.tid \\
\qquad\qquad\qquad\ \ top \triangleq Head(thrds[here][tid].stack) \\
\qquad\qquad\qquad\ \ blk \triangleq top.b \\
\qquad\qquad\ \ \text{IN}\ \ \wedge\ msg \neq C!NotMessage \\
\qquad\qquad\qquad\ \ \wedge\ SetActionNameAndDepth(\langle \text{“ReleaseRootFinish”},\ here \rangle) \\
\qquad\qquad\qquad\ \ \wedge\ C!RecvMsg(msg) \\
\qquad\qquad\qquad\ \ \wedge\ fstates' = [fstates\ \text{EXCEPT}\ ![root].status = \text{“forgotten”}] \\
\qquad\qquad\qquad\ \ \wedge\ waitForMsgs' = waitForMsgs \setminus \{[src \mapsto here, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad dst \mapsto here, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad target \mapsto here, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad fid \mapsto root, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad type \mapsto \text{“masterCompletedDone”}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad finishEnd \mapsto \text{TRUE}\ ], \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad [src \mapsto here, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad dst \mapsto here, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad target \mapsto here, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad fid \mapsto root, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad type \mapsto \text{“masterCompletedDone”}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad finishEnd \mapsto \text{FALSE}\ ]\} \\
\qquad\qquad\qquad\ \ \wedge\ UnblockStopFinish(here,\ tid,\ root,\ blk) \\
\quad \wedge \text{UNCHANGED}\ \langle convertSet,\ adoptSet,\ mastersStatus,\ seq, \\
\qquad\qquad\qquad\ killed,\ pendingAct,\ \ fmasters,\ fbackups \rangle
\end{array}
$$

Finish master replica actions

$$
\begin{array}{l}
MasterTransit\ \triangleq \\
\quad \wedge\ pstate = \text{“running”} \\
\quad \wedge\ msgs \neq \{\} \\
\quad \wedge \text{LET}\ msg \triangleq C!FindMSG(\text{“masterTransit”}) \\
\quad\ \ \text{IN}\ \ \wedge\ msg \neq C!NotMessage
\end{array}
$$

$\land$ LET $here \triangleq msg.dst$
$fid \triangleq msg.fid$
$src \triangleq msg.src$
$target \triangleq msg.target$
$backupPlace \triangleq C!GetBackup(here)$
IN $\land SetActionNameAndDepth(\langle$"MasterTransit", $here\rangle)$
$\land mastersStatus[here].status =$ "running"
$\land src \neq C!NotPlace$
$\land fid \neq C!NotID$
$\land fstates[fid].here = here$
$\land$ 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 $![fid].transit[src][target] = @ + 1,$
$![fid].numActive = @ + 1]$
$\land C!ReplaceMsg(msg, [mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto backupPlace,$
$source \mapsto src,$
$target \mapsto target,$
$fid \mapsto fid,$
$type \mapsto$ "backupTransit"$])$
$\land C!IncrMSEQ(1)$
$\land waitForMsgs' = waitForMsgs \cup \{[src \mapsto backupPlace,$
$dst \mapsto here,$
$fid \mapsto fid,$
$source \mapsto src,$
$target \mapsto target,$
$type \mapsto$ "backupTransitDone" $]\}$
$\land$ UNCHANGED $\langle convertSet, adoptSet, mastersStatus, fstates, pstate,$
$thrds, killed, pendingAct, fbackups, blockedThrds, runningThrds\rangle$

$MasterLive \triangleq$
$\land pstate =$ "running"
$\land msgs \neq \{\}$
$\land$ LET $msg \triangleq C!FindMSG($"masterLive"$)$
IN $\land msg \neq C!NotMessage$
$\land$ LET $here \triangleq msg.dst$
$fid \triangleq msg.fid$
$backupPlace \triangleq C!GetBackup(here)$
IN $\land SetActionNameAndDepth(\langle$"MasterLive", $here\rangle)$
$\land fid \neq C!NotID$

$\land\ backupPlace \neq C!NotPlace$
$\land\ fstates[fid].here = here$
$\land\ mastersStatus[here].status = \text{“running”}$
$\land$ LET $source \triangleq msg.source$
$target \triangleq msg.target$
$submit \triangleq \neg(source \in killed \lor target \in killed)$
IN IF $submit$
THEN $\land\ fmasters[fid].transit[source][target] > 0$
$\land\ fmasters' = [fmasters$ EXCEPT $![fid].transit[source][target] = @ - 1,$
$![fid].live[target] = @ + 1]$
$\land\ C!ReplaceMsg(msg,$
$[\ mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto backupPlace,$
$source \mapsto source,$
$target \mapsto target,$
$fid \mapsto fid,$
$aid \mapsto msg.aid,$
$type \mapsto \text{“backupLive”}])$
$\land\ C!IncrMSEQ(1)$
$\land\ waitForMsgs' = waitForMsgs \cup \{[src \mapsto backupPlace,$
$dst \mapsto here,$
$fid \mapsto fid,$
$source \mapsto source,$
$target \mapsto target,$
$aid \mapsto msg.aid,$
$type \mapsto \text{“backupLiveDone”}\ ]\}$
ELSE $\land\ fmasters' = fmasters$
$\land\ waitForMsgs' = waitForMsgs$
$\land\ C!ReplaceMsg(msg,$
$[\ mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto target,$
$source \mapsto source,$
$target \mapsto target,$
$fid \mapsto fid,$
$aid \mapsto msg.aid,$
$type \mapsto \text{“masterLiveDone”},$
$isAdopted \mapsto$ FALSE,
$adoptedRoot \mapsto C!NotID,$
$submit \mapsto$ FALSE,
$success \mapsto$ TRUE$])$
$\land\ C!IncrMSEQ(1)$
$\land$ UNCHANGED $\langle convertSet, adoptSet, mastersStatus,$
$fstates, pstate, thrds, killed, pendingAct, fbackups,$

$blockedThrds,\ runningThrds\rangle$

$$
\begin{array}{l}
MasterCompleted \triangleq \\
\quad \land\ pstate = \text{“running”} \\
\quad \land\ msgs \neq \{\} \\
\quad \land\ \text{LET}\ msg \triangleq C!FindMSG(\text{“masterCompleted”}) \\
\qquad \text{IN} \quad \land\ msg \neq C!NotMessage \\
\qquad\qquad \land\ \text{LET}\ here \triangleq msg.dst \\
\qquad\qquad\qquad\quad mid \triangleq msg.mid \\
\qquad\qquad\qquad\quad fid \triangleq msg.fid \\
\qquad\qquad\qquad\quad src \triangleq msg.src \\
\qquad\qquad\qquad\quad target \triangleq msg.target \\
\qquad\qquad\qquad\quad backupPlace \triangleq C!GetBackup(here) \\
\qquad\qquad\qquad\quad finishEnd \triangleq msg.finishEnd \\
\qquad\qquad\ \text{IN} \quad \land\ SetActionNameAndDepth(\langle \text{“MasterCompleted”}, here\rangle) \\
\qquad\qquad\qquad\quad \land\ backupPlace \neq C!NotPlace \\
\qquad\qquad\qquad\quad \land\ fid \neq C!NotID \\
\qquad\qquad\qquad\quad \land\ fstates[fid].here = here \\
\qquad\qquad\qquad\quad \land\ mastersStatus[here].status = \text{“running”} \\
\qquad\qquad\qquad\quad \land\ fmasters[fid].live[target] \quad > 0 \\
\qquad\qquad\qquad\quad \land\ fmasters[fid].numActive > 0 \\
\qquad\qquad\qquad\quad \land\ fmasters' = [fmasters\ \text{EXCEPT}\ ![fid].live[target] = @ - 1, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad ![fid].numActive = @ - 1] \\
\qquad\qquad\qquad\quad \land\ C!ReplaceMsg(msg, \\
\qquad\qquad\qquad\qquad\qquad [mid \mapsto seq.mseq, \\
\qquad\qquad\qquad\qquad\qquad\ \ src \mapsto here, \\
\qquad\qquad\qquad\qquad\qquad\ \ dst \mapsto backupPlace, \\
\qquad\qquad\qquad\qquad\qquad target \mapsto target, \\
\qquad\qquad\qquad\qquad\qquad\ \ fid \mapsto fid, \\
\qquad\qquad\qquad\qquad finishEnd \mapsto finishEnd, \\
\qquad\qquad\qquad\qquad\qquad\ \ type \mapsto \text{“backupCompleted”}]) \\
\qquad\qquad\qquad\quad \land\ C!IncrMSEQ(1) \\
\qquad\qquad\qquad\quad \land\ waitForMsgs' = waitForMsgs \cup \{[src \quad \mapsto backupPlace, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad dst \quad \mapsto here, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad fid \quad \mapsto fid, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad target \quad \mapsto target, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad finishEnd \mapsto finishEnd, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad type \mapsto \text{“backupCompletedDone”}\ ]\} \\
\quad \land\ \text{UNCHANGED}\ \langle convertSet, adoptSet, mastersStatus, fstates, pstate, \\
\qquad\qquad\qquad thrds, killed, pendingAct, fbackups, blockedThrds, runningThrds\rangle
\end{array}
$$

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)
          IN  ∧ SetActionNameAndDepth(⟨"AdopterTransit", here⟩)
              ∧ mastersStatus[here].status = "running"
              ∧ fid ≠ C!NotID
              ∧ fstates[fid].here = here    FIXME: the code doesn't check if src or target at alive!!!!
              ∧ fmasters' = [fmasters EXCEPT ![fid].transitAdopter[src][target] = @ + 1,
                                             ![fid].numActive = @ + 1]
              ∧ C!ReplaceMsg(msg,
                             [ mid ↦ seq.mseq,
                               src ↦ here,
                               dst ↦ src,
                            target ↦ target,
                               fid ↦ fid,
                              type ↦ "masterTransitDone",
                        isAdopted ↦ FALSE,
                       adoptedRoot ↦ C!NotID,
                       adoptedFID ↦ C!NotID,
                          success ↦ TRUE])
              ∧ C!IncrMSEQ(1)
  ∧ UNCHANGED ⟨convertSet, adoptSet, mastersStatus, fstates, pstate,
               thrds, killed, pendingAct, fbackups,
               blockedThrds, runningThrds⟩

AdopterLive ≜
  ∧ pstate = "running"
  ∧ msgs ≠ {}
  ∧ LET msg ≜ C!FindMSG("adopterLive")
    IN  ∧ msg ≠ C!NotMessage
        ∧ msg.dst ∉ killed
        ∧ LET here ≜ msg.dst
                fid ≜ msg.fid
                backupPlace ≜ C!GetBackup(here)
          IN  ∧ SetActionNameAndDepth(⟨"AdopterLive", here⟩)
              ∧ fid ≠ C!NotID
              ∧ backupPlace ≠ C!NotPlace
              ∧ fstates[fid].here = here
```

$\land$ $masters Status[here].status = \text{“running”}$
$\land$ LET $source \triangleq msg.source$
$target \triangleq msg.target$
$submit \triangleq \neg(source \in killed \lor target \in killed)$
IN $\land$ IF $submit$
THEN $\land fmasters[fid].transitAdopted[source][target] > 0$
$\land fmasters' = [fmasters$ EXCEPT $![fid].transitAdopted[source][target] = @ -$
$![fid].liveAdopted[target] = @ + 1]$
ELSE $fmasters' = fmasters$
$\land C!ReplaceMsg(msg,$
$[\ mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto target,$
$source \mapsto source,$
$target \mapsto target,$
$fid \mapsto fid,$
$aid \mapsto msg.aid,$
$type \mapsto \text{“masterLiveDone”},$
$isAdopted \mapsto$ FALSE,
$adoptedRoot \mapsto C!NotID,$
$submit \mapsto submit,$
$success \mapsto$ TRUE$])$
$\land C!IncrMSEQ(1)$
$\land$ UNCHANGED $\langle convertSet, adoptSet, mastersStatus, fstates, pstate, waitForMsgs,$
$thrds, killed, pendingAct, fbackups, blockedThrds, runningThrds\rangle$

$AdopterCompleted \triangleq$
$\land pstate = \text{“running”}$
$\land msgs \neq \{\}$
$\land$ LET $msg \triangleq C!FindMSG(\text{“adopterCompleted”})$
IN $\land msg \neq C!NotMessage$
$\land msg.dst \notin killed$
$\land$ 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 $\land SetActionNameAndDepth(\langle \text{“AdopterCompleted”}, here\rangle)$
$\land mastersStatus[here].status = \text{“running”}$
$\land backupPlace \neq C!NotPlace$
$\land fid \neq C!NotID$
$\land fstates[fid].here = here$
$\land fmasters[fid].liveAdopted[target] > 0$

$\land\ fmasters[fid].numActive > 0$
$\land\ fmasters' = [fmasters \text{ EXCEPT } ![fid].liveAdopted[target] = @ - 1,$
$\qquad\qquad ![fid].numActive = @ - 1]$
$\land$ IF $fmasters'[fid].numActive = 0$
THEN $\land\ C!ReplaceMsgSet(msg, \{$
$[\ mid \mapsto seq.mseq,$
$\ src \mapsto here,$
$\ dst \mapsto target,$
$\ target \mapsto target,$
$\ fid \mapsto fid,$
$\ type \mapsto$ "masterCompletedDone",
$\ finishEnd \mapsto finishEnd,$
$\ isAdopted \mapsto$ FALSE,
$\ adoptedRoot \mapsto C!NotID,$
$\ numActive \mapsto fmasters[fid].numActive,$
$\ success \mapsto$ TRUE],
$[mid \mapsto seq.mseq + 1,$
$\ src \mapsto here,$
$\ dst \mapsto here,$
$\ fid \mapsto fid,$
$\ type \mapsto$ "releaseFinish"]$\})$
$\land\ C!IncrMSEQ(2)$
ELSE $\land\ C!ReplaceMsg(msg,$
$[\ mid \mapsto seq.mseq,$
$\ src \mapsto here,$
$\ dst \mapsto target,$
$\ target \mapsto target,$
$\ fid \mapsto fid,$
$\ numActive \mapsto fmasters[fid].numActive,$
$\ isAdopted \mapsto$ FALSE,
$\ adoptedRoot \mapsto C!NotID,$
$\ type \mapsto$ "masterCompletedDone",
$\ finishEnd \mapsto finishEnd,$
$\ success \mapsto$ TRUE$])$
$\land\ C!IncrMSEQ(1)$
$\land$ UNCHANGED $\langle convertSet, adoptSet, mastersStatus,$
$fstates, pstate, thrds, killed, pendingAct, fbackups,$
$waitForMsgs, blockedThrds, runningThrds\rangle$

Finish backup replica actions

$BackupTransit \triangleq$
$\land\ pstate =$ "running"
$\land\ msgs \neq \{\}$
$\land$ LET $msg \triangleq C!FindMSG($"backupTransit"$)$
IN $\land\ msg \neq C!NotMessage$

$$
\begin{array}{l}
\land \text{LET } here \triangleq msg.dst \\
\quad fid \triangleq msg.fid \\
\quad src \triangleq msg.src \\
\quad source \triangleq msg.source \\
\quad target \triangleq msg.target \\
\quad type \triangleq \text{IF } src = fstates[fid].here \\
\qquad\qquad \text{THEN } \text{“backupTransitDone”} \\
\qquad\qquad \text{ELSE } \text{“masterTransitDone”} \\
\text{IN} \quad \land SetActionNameAndDepth(\langle \text{“BackupTransit”}, here \rangle) \\
\quad \land fmasters[fid].backupPlace = here \\
\quad \land \text{IF } fbackups[fid].isAdopted \\
\qquad \text{THEN} \quad \land fbackups' = fbackups \\
\qquad\qquad \land C!ReplaceMsg(msg, \\
\qquad\qquad\qquad [\ mid \mapsto seq.mseq, \\
\qquad\qquad\qquad\ \ src \mapsto here, \\
\qquad\qquad\qquad\ \ dst \mapsto src, \\
\qquad\qquad\qquad\ \ target \mapsto target, \\
\qquad\qquad\qquad\ \ source \mapsto source, \\
\qquad\qquad\qquad\ \ fid \mapsto fid, \\
\qquad\qquad\qquad\ \ type \mapsto type, \\
\qquad\qquad\qquad\ \ isAdopted \mapsto \text{TRUE}, \\
\qquad\qquad\qquad\ \ adoptedRoot \mapsto fbackups[fid].adoptedRoot, \\
\qquad\qquad\qquad\ \ adoptedFID \mapsto fid, \\
\qquad\qquad\qquad\ \ success \mapsto \text{FALSE}]) \\
\qquad\qquad \land C!IncrMSEQ(1) \\
\qquad \text{ELSE} \quad \land \text{IF } fbackups[fid].id = C!NotID \\
\qquad\qquad\ \ \text{THEN } fbackups' = [fbackups \text{ EXCEPT } ![fid].id = fid, \\
\qquad\qquad\qquad\qquad\qquad\qquad ![fid].transit[source][target] = @ + 1, \\
\qquad\qquad\qquad\qquad\qquad\qquad ![fid].live[src] = 1, \\
\qquad\qquad\qquad\qquad\qquad\qquad ![fid].numActive = @ + 2] \\
\qquad\qquad\ \ \text{ELSE } fbackups' = [fbackups \text{ EXCEPT } ![fid].transit[source][target] = @ + 1, \\
\qquad\qquad\qquad\qquad\qquad\qquad ![fid].numActive = @ + 1] \\
\qquad\qquad \land C!ReplaceMsg(msg, \\
\qquad\qquad\qquad [\ mid \mapsto seq.mseq, \\
\qquad\qquad\qquad\ \ src \mapsto here, \\
\qquad\qquad\qquad\ \ dst \mapsto src, \\
\qquad\qquad\qquad\ \ target \mapsto target, \\
\qquad\qquad\qquad\ \ source \mapsto source, \\
\qquad\qquad\qquad\ \ fid \mapsto fid, \\
\qquad\qquad\qquad\ \ type \mapsto type, \\
\qquad\qquad\qquad\ \ isAdopted \mapsto \text{FALSE}, \\
\qquad\qquad\qquad\ \ adoptedRoot \mapsto C!NotID, \\
\qquad\qquad\qquad\ \ adoptedFID \mapsto C!NotID, \\
\qquad\qquad\qquad\ \ success \mapsto \text{TRUE}]) \\
\qquad\qquad \land C!IncrMSEQ(1)
\end{array}
$$

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

$BackupLive \triangleq$
$\land\ pstate = \text{“running”}$
$\land\ msgs \neq \{\}$
$\land$ LET $msg \triangleq C!FindMSG(\text{“backupLive”})$
IN $\land\ msg \neq C!NotMessage$
$\land$ LET $here \triangleq msg.dst$
$fid \triangleq msg.fid$
$src \triangleq msg.src$
$source \triangleq msg.source$
$target \triangleq msg.target$
$type \triangleq$ IF $src = fstates[fid].here$
THEN “backupLiveDone”
ELSE “masterLiveDone”
IN $\land\ SetActionNameAndDepth(\langle \text{“BackupLive”}, here\rangle)$
$\land\ fmasters[fid].backupPlace = here$
$\land$ IF $fbackups[fid].isAdopted$
THEN $\land\ fbackups' = fbackups$
$\land\ C!ReplaceMsg(msg,$
$[\ mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto src,$
$target \mapsto target,$
$source \mapsto source,$
$fid \mapsto fid,$
$aid \mapsto msg.aid,$
$type \mapsto type,$
$isAdopted \mapsto$ TRUE,
$adoptedRoot \mapsto fbackups[fid].adoptedRoot,$
$success \mapsto$ FALSE,
$submit \mapsto$ FALSE$])$
$\land\ C!IncrMSEQ(1)$
ELSE $\land\ fbackups[fid].transit[source][target] > 0$
$\land\ fbackups' = [fbackups$ EXCEPT $![fid].transit[source][target] = @ - 1,$
$![fid].live[target] = @ + 1]$
$\land\ C!ReplaceMsg(msg,$
$[\ mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto src,$
$target \mapsto target,$
$source \mapsto source,$
$fid \mapsto fid,$

$$
\begin{array}{rl}
& aid \mapsto msg.aid, \\
& type \mapsto type, \\
& isAdopted \mapsto \text{FALSE}, \\
& adoptedRoot \mapsto C!NotID, \\
& success \mapsto \text{TRUE}, \\
& submit \mapsto \text{TRUE}])
\end{array}
$$

$$
\begin{array}{l}
\quad \wedge\ C!IncrMSEQ(1) \\
\wedge\ \text{UNCHANGED}\ \langle convertSet,\ fstates,\ pstate,\ thrds,\ pendingAct,\ fmasters,\ waitForMsgs, \\
\qquad\qquad blockedThrds,\ runningThrds,\ killed,\ \ adoptSet,\ mastersStatus\rangle
\end{array}
$$

$$
\begin{array}{l}
BackupCompleted\ \triangleq \\
\quad \wedge\ pstate = \text{“running”} \\
\quad \wedge\ msgs \neq \{\} \\
\quad \wedge\ \text{LET}\ msg\ \triangleq\ C!FindMSG(\text{“backupCompleted”}) \\
\qquad \text{IN} \quad \wedge\ msg \neq C!NotMessage \\
\qquad\qquad \wedge\ \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 type\ \triangleq\ \text{IF}\ src = fstates[fid].here \\
\qquad\qquad\qquad\qquad\qquad \text{THEN}\ \text{“backupCompletedDone”} \\
\qquad\qquad\qquad\qquad\qquad \text{ELSE}\ \text{“masterCompletedDone”} \\
\qquad\qquad\qquad finishEnd\ \triangleq\ msg.finishEnd \\
\qquad\qquad\ \text{IN} \quad \wedge\ fmasters[fid].backupPlace = here \\
\qquad\qquad\qquad \wedge\ SetActionNameAndDepth(\langle \text{“BackupCompleted”},\ here\rangle) \\
\qquad\qquad\qquad \wedge\ \text{IF}\ fbackups[fid].isAdopted \\
\qquad\qquad\qquad\quad \text{THEN}\ \wedge\ fbackups' = fbackups \\
\qquad\qquad\qquad\qquad\qquad \wedge\ C!ReplaceMsg(msg, \\
\qquad\qquad\qquad\qquad\qquad\qquad [\ mid \mapsto seq.mseq, \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \ src \mapsto here, \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \ dst \mapsto src, \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \ target \mapsto target, \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \ numActive \mapsto 1000, \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \ fid \mapsto fid, \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \ type \mapsto type, \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \ isAdopted \mapsto \text{TRUE}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \ adoptedRoot \mapsto fbackups[fid].adoptedRoot, \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \ finishEnd \mapsto finishEnd, \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \ success \mapsto \text{FALSE}]) \\
\qquad\qquad\qquad\qquad\qquad \wedge\ C!IncrMSEQ(1) \\
\qquad\qquad\qquad\quad \text{ELSE}\ \wedge\ fbackups[fid].live[target] > 0 \\
\qquad\qquad\qquad\qquad\qquad \wedge\ fbackups[fid].numActive > 0 \\
\qquad\qquad\qquad\qquad\qquad \wedge\ fbackups' = [fbackups\ \text{EXCEPT}\ ![fid].live[target] = @ - 1, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \ ![fid].numActive = @ - 1] \\
\qquad\qquad\qquad\qquad\qquad \wedge\ C!ReplaceMsg(msg,
\end{array}
$$

$$\begin{array}{rl}
[\ mid & \mapsto seq.mseq, \\
src & \mapsto here, \\
dst & \mapsto src, \\
target & \mapsto target, \\
numActive & \mapsto 1000, \\
fid & \mapsto fid, \\
type & \mapsto type, \\
finishEnd & \mapsto finishEnd, \\
success & \mapsto \text{TRUE}])
\end{array}$$

$$\wedge\ C!IncrMSEQ(1)$$

$$\wedge\ \text{UNCHANGED}\ \langle convertSet,\ adoptSet,\ mastersStatus,\ fstates,\ pstate,\ thrds,\ killed,\ pendingAct,\ fmasters,\ waitForMsgs,\ blockedThrds,\ runningThrds\rangle$$

Finish adoption actions for recovery

$$GetAdoptionSeeker \triangleq$$
$$\text{IF}\ adoptSet = \{\}\ \text{THEN}\ C!NotAdopter$$
$$\text{ELSE}\ \text{CHOOSE}\ m \in adoptSet : mastersStatus[m.here].status = \text{“seekAdoption”}$$

$$SeekAdoption \triangleq$$
$$\wedge\ pstate = \text{“running”}$$
$$\wedge\ \exists\, p \in PLACE : mastersStatus[p].status = \text{“seekAdoption”}$$
$$\wedge\ \text{LET}\ pair \triangleq GetAdoptionSeeker$$
$$\text{IN}\quad \wedge\ pair \neq C!NotAdopter$$
$$\wedge\ pair.here \notin killed$$
$$\wedge\ \text{LET}\ here \triangleq pair.here$$
$$adopter \triangleq pair.adopter$$
$$child \triangleq pair.child$$
$$\text{IN}\quad \wedge\ SetActionNameAndDepth(\langle \text{“SeekAdoption”}, here\rangle)$$
$$\wedge\ fbackups' = [fbackups\ \text{EXCEPT}\ ![child].isAdopted = \text{TRUE},$$
$$![child].adoptedRoot = adopter]$$
$$\wedge\ fmasters' = [fmasters\ \text{EXCEPT}\ ![adopter].children = fmasters[adopter].children \setminus \{child\},$$
$$![adopter].liveAdopted =$$
$$[p \in PLACE \mapsto fmasters[adopter].liveAdopted[p]$$
$$+ fbackups[child].live[p]],$$
$$![adopter].transitAdopted =$$
$$[p \in PLACE \mapsto$$
$$[q \in PLACE \mapsto fmasters[adopter].transitAdopted[p][q]$$
$$+ fbackups[child].transit[p][q]]],$$
$$![adopter].numActive = @ + fbackups[child].numActive]$$
$$\wedge\ adoptSet' = adoptSet \setminus \{pair\}$$
$$\wedge\ \text{IF}\ \exists\, m \in adoptSet' : m.here = here$$
$$\text{THEN}\ \wedge\ mastersStatus' = mastersStatus$$
$$\text{ELSE}\ \wedge\ mastersStatus' = [mastersStatus\ \text{EXCEPT}\ ![here].status = \text{“convertDead”}]$$

$\land$ UNCHANGED $\langle fstates,\ msgs,\ pstate,\ seq,\ thrds,\ killed,\ pendingAct,\ \ waitForMsgs,$
$convertSet,\ blockedThrds,\ runningThrds\rangle$

$GetConvertSeeker \triangleq$
IF $convertSet = \{\}$ THEN $C!NotConvTask$
ELSE CHOOSE $m \in convertSet : mastersStatus[m.here].status =$ "convertDead"

$ConvertDeadActivities \triangleq$
$\land\ pstate =$ "running"
$\land\ \exists\, p \in PLACE : mastersStatus[p].status =$ "convertDead"
$\land$ LET $convSeeker \triangleq GetConvertSeeker$
IN $\land\ convSeeker \neq C!NotConvTask$
$\land\ convSeeker.here \notin killed$
$\land$ LET $here \triangleq convSeeker.here$
$pl \triangleq convSeeker.pl$
$fid \triangleq convSeeker.fid$
$dead \triangleq mastersStatus[here].lastKilled$
IN $\land\ SetActionNameAndDepth(\langle$"ConvertDeadActivities", $here\rangle)$
$\land\ convSet' = convSet \setminus \{convSeeker\}$
$\land\ fmasters[fid].transitAdopted[pl][dead] \geq 0$
$\land\ fmasters[fid].transitAdopted[dead][pl] \geq 0$
$\land\ fmasters[fid].liveAdopted[dead] \geq 0$
$\land\ fmasters' = [fmasters$ EXCEPT $![fid].numActive =$
@ $- fmasters[fid].transit[pl][dead]$
$- fmasters[fid].transit[dead][pl]$
$- fmasters[fid].transitAdopted[pl][dead]$
$- fmasters[fid].transitAdopted[dead][pl]$
$- fmasters[fid].live[dead]$
$- fmasters[fid].liveAdopted[dead],$
$![fid].transit[pl][dead] = 0,$
$![fid].transitAdopted[pl][dead] = 0,$
$![fid].transit[dead][pl] = 0,$
$![fid].transitAdopted[dead][pl] = 0,$
$![fid].live[dead] = 0,$
$![fid].liveAdopted[dead] = 0]$
$\land$ IF $fmasters'[fid].numActive = 0$
THEN $\land\ C!SendMsg([mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto here,$
$fid \mapsto fid,$
$type \mapsto$ "releaseFinish"$])$
$\land\ C!IncrMSEQ(1)$
ELSE $\land\ msgs' = msgs$
$\land\ seq' = seq$

$\land$ IF $\exists\, m \in convertSet' : m.here = here$
THEN $mastersStatus' = mastersStatus$
ELSE $mastersStatus' = [mastersStatus$ EXCEPT $![here].status =$ "running"$]$
$\land$ UNCHANGED $\langle fstates,\ pstate,\ thrds,\ killed,\ pendingAct,\ fbackups,\ waitForMsgs,$
$adoptSet,\ blockedThrds,\ runningThrds\rangle$

$FindWaitForMSG \triangleq$
LET $mset \triangleq \{m \in waitForMsgs :$
$\land\, m.src \in killed$
$\land\, m.dst \notin killed$
$\land\, m.src \in killed\}$
IN IF $mset = \{\}$ THEN $C!NotMessage$
ELSE CHOOSE $x \in mset :$ TRUE

$SimulateFailedResponse \triangleq$
$\land\, pstate =$ "running"
$\land\, killed \neq \{\}$
$\land\, waitForMsgs \neq \{\}$
$\land$ LET $msg \triangleq FindWaitForMSG$
IN $\land\, msg \neq C!NotMessage$
$\land$ LET $dead \triangleq msg.src$
$here \triangleq msg.dst$
$delMsgs \triangleq \{m \in msgs : m.dst = dead\ \}$
$wfm \triangleq \{m \in waitForMsgs : m.dst = dead\}$
IN $\land\, SetActionNameAndDepth(\langle$"SimulateFailedResponse", $here\rangle)$
$\land\, waitForMsgs' = (waitForMsgs \setminus wfm) \setminus \{msg\}$
$\land\, C!IncrMSEQ(1)$
$\land$ IF $msg.type =$ "masterLiveDone"
THEN IF $\neg(\exists\, m \in msgs :$ message has been sent already
$\land\, m.type = msg.type \land m.src = msg.src$
$\land\, m.dst = msg.dst \land m.fid = msg.fid$
$\land\, m.aid = msg.aid \land m.success)$
THEN $\land\, msgs' = (msgs \setminus delMsgs) \cup \{$
$[\ mid \mapsto seq.mseq,$
$src \mapsto msg.src,$
$dst \mapsto msg.dst,$
$source \mapsto msg.source,$
$target \mapsto msg.dst,$
$fid \mapsto msg.fid,$
$aid \mapsto msg.aid,$
$type \mapsto$ "masterLiveDone",
$isAdopted \mapsto$ FALSE,
$adoptedRoot \mapsto C!NotID,$
$submit \mapsto$ FALSE,

$$
\begin{array}{l}
\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{“masterCompletedDone”} \\
\text{THEN IF } \neg(\exists\, m \quad \in msgs : \ \text{message has been sent already} \\
\qquad\qquad\qquad\quad \wedge\, m.type = msg.type \wedge m.src = msg.src \\
\qquad\qquad\qquad\quad \wedge\, m.dst = msg.dst \wedge m.fid = msg.fid \\
\qquad\qquad\qquad\quad \wedge\, m.success) \\
\qquad\quad \text{THEN} \quad \wedge\, msgs' = (msgs \setminus delMsgs) \cup \{ \\
\qquad\qquad\qquad\qquad [\ mid \quad \mapsto seq.mseq, \\
\qquad\qquad\qquad\qquad\ \ src \quad \mapsto msg.src, \\
\qquad\qquad\qquad\qquad\ \ dst \quad \mapsto msg.dst, \\
\qquad\qquad\qquad\qquad target \quad \mapsto msg.target, \\
\qquad\qquad\qquad\qquad\ \ fid \quad \mapsto msg.fid, \\
\qquad\qquad\qquad\quad numActive \mapsto 1000, \\
\qquad\qquad\qquad\quad isAdopted \ \mapsto \text{FALSE}, \\
\qquad\qquad\qquad adoptedRoot \ \mapsto C!NotID, \\
\qquad\qquad\qquad\qquad type \quad \mapsto \text{“masterCompletedDone”}, \\
\qquad\qquad\qquad finishEnd \quad \mapsto msg.finishEnd, \\
\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{“masterTransitDone”} \\
\text{THEN IF } \neg(\exists\, m \in msgs : \ \text{message has been sent already} \\
\qquad\qquad\qquad\quad \wedge\, m.type = msg.type \wedge m.src = msg.src \\
\qquad\qquad\qquad\quad \wedge\, m.dst = msg.dst \wedge m.fid = msg.fid \\
\qquad\qquad\qquad\quad \wedge\, m.success) \\
\qquad\quad \text{THEN} \quad \wedge\, msgs' = (msgs \setminus delMsgs) \cup \{ \\
\qquad\qquad\qquad\qquad [\ mid \quad \mapsto seq.mseq, \\
\qquad\qquad\qquad\qquad\ \ src \quad \mapsto msg.src, \\
\qquad\qquad\qquad\qquad\ \ dst \quad \mapsto msg.dst, \\
\qquad\qquad\qquad\qquad target \quad \mapsto msg.target, \\
\qquad\qquad\qquad\qquad\ \ fid \quad \mapsto msg.fid, \\
\qquad\qquad\qquad\qquad type \quad \mapsto \text{“masterTransitDone”}, \\
\qquad\qquad\qquad\quad isAdopted \mapsto \text{FALSE}, \\
\qquad\qquad\qquad adoptedRoot \ \mapsto C!NotID, \\
\qquad\qquad\qquad adoptedFID \ \mapsto C!NotID, \\
\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\quad \wedge\, m.type = msg.type \wedge m.src = msg.src \\
\qquad\qquad\qquad\quad \wedge\, m.dst = msg.dst \wedge m.fid = msg.fid \\
\qquad\qquad\qquad\quad \wedge\, m.success) \\
\qquad\quad \text{THEN} \quad \wedge\, msgs' = (msgs \setminus delMsgs) \cup \{ \\
\qquad\qquad\qquad\qquad [\ mid \quad \mapsto seq.mseq, \\
\qquad\qquad\qquad\qquad\ \ src \quad \mapsto msg.src,
\end{array}
$$

$$
\begin{array}{rl}
 & \quad dst \mapsto msg.dst, \\
 & \quad target \mapsto msg.target, \\
 & \quad numActive \mapsto 1000, \\
 & \quad fid \mapsto msg.fid, \\
 & \quad type \mapsto \text{“backupCompletedDone”}, \\
 & \quad finishEnd \mapsto msg.finishEnd, \\
 & \quad success \mapsto \text{FALSE}]\} \\
\text{ELSE} & \wedge msgs' = (msgs \setminus delMsgs) \\
\text{ELSE IF} & msg.type = \text{“backupLiveDone”} \\
\text{THEN IF} & \neg(\exists\, m \in msgs : \text{message has been sent already} \\
 & \wedge m.type = msg.type \wedge m.src = msg.src \\
 & \wedge m.dst = msg.dst \wedge m.fid = msg.fid \\
 & \wedge m.source = msg.source \wedge m.success) \\
\text{THEN} & \wedge msgs' = (msgs \setminus delMsgs) \cup \{ \\
 & \quad [mid \mapsto seq.mseq, \\
 & \quad src \mapsto msg.src, \\
 & \quad dst \mapsto msg.dst, \\
 & \quad target \mapsto msg.target, \\
 & \quad source \mapsto msg.source, \\
 & \quad fid \mapsto msg.fid, \\
 & \quad aid \mapsto msg.aid, \\
 & \quad type \mapsto \text{“backupLiveDone”}, \\
 & \quad isAdopted \mapsto \text{FALSE}, \\
 & \quad adoptedRoot \mapsto C!NotID, \\
 & \quad success \mapsto \text{FALSE}, \\
 & \quad submit \mapsto \text{FALSE}]\} \\
\text{ELSE} & \wedge msgs' = (msgs \setminus delMsgs) \\
\text{ELSE IF} & msg.type = \text{“backupTransitDone”} \\
\text{THEN IF} & \neg(\exists\, m \in msgs : \text{message has been sent already} \\
 & \wedge m.type = msg.type \wedge m.src = msg.src \\
 & \wedge m.dst = msg.dst \wedge m.fid = msg.fid \\
 & \wedge m.target = msg.target \wedge m.success) \\
\text{THEN} & \wedge msgs' = (msgs \setminus delMsgs) \cup \{ \\
 & \quad [mid \mapsto seq.mseq, \\
 & \quad src \mapsto msg.src, \\
 & \quad dst \mapsto msg.dst, \\
 & \quad target \mapsto msg.target, \\
 & \quad source \mapsto msg.source, \\
 & \quad fid \mapsto msg.fid, \\
 & \quad type \mapsto \text{“backupTransitDone”}, \\
 & \quad isAdopted \mapsto \text{FALSE}, \\
 & \quad adoptedRoot \mapsto C!NotID, \\
 & \quad adoptedFID \mapsto C!NotID, \\
 & \quad success \mapsto \text{FALSE}]\} \\
\text{ELSE} & \wedge msgs' = (msgs \setminus delMsgs)
\end{array}
$$

ELSE FALSE

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

Predicate enumerating all possible next actions

$Next \triangleq$

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

Asserting fairness properties to all actions

$Liveness \triangleq$

- $\land$ $\mathrm{WF}_{Vars}(RecvAsync)$
- $\land$ $\mathrm{WF}_{Vars}(ReleaseRootFinish)$
- $\land$ $\mathrm{WF}_{Vars}(AuthorizeReceivedAsync)$
- $\land$ $\mathrm{WF}_{Vars}(StartFinish)$
- $\land$ $\mathrm{WF}_{Vars}(StopFinish)$
- $\land$ $\mathrm{WF}_{Vars}(SpawnLocalAsync)$
- $\land$ $\mathrm{WF}_{Vars}(SpawnRemoteAsync)$

$\land \text{WF}_{Vars}(TerminateAsync)$
$\land \text{WF}_{Vars}(ScheduleNestedFinish)$
$\land \text{WF}_{Vars}(RunExprOrKill)$
$\land \text{WF}_{Vars}(BackupTransit)$
$\land \text{WF}_{Vars}(BackupLive)$
$\land \text{WF}_{Vars}(BackupCompleted)$
$\land \text{WF}_{Vars}(MasterTransit)$
$\land \text{WF}_{Vars}(MasterLive)$
$\land \text{WF}_{Vars}(MasterCompleted)$
$\land \text{WF}_{Vars}(MasterTransitDone)$
$\land \text{WF}_{Vars}(MasterLiveDone)$
$\land \text{WF}_{Vars}(MasterCompletedDone)$
$\land \text{WF}_{Vars}(AdopterTransit)$
$\land \text{WF}_{Vars}(AdopterLive)$
$\land \text{WF}_{Vars}(AdopterCompleted)$
$\land \text{WF}_{Vars}(SeekAdoption)$
$\land \text{WF}_{Vars}(ConvertDeadActivities)$
$\land \text{WF}_{Vars}(SimulateFailedResponse)$
$\land \text{WF}_{Vars}(AuthorizeTransitAsync)$
$\land \text{WF}_{Vars}(UnblockTerminateAsync)$

Specification

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

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

* Modification History
* Last modified *Mon Dec* 18 10:24:26 *AEDT* 2017 by *u5482878*
* Last modified *Tue Dec* 05 18:31:58 *AEDT* 2017 by *shamouda*
* Created *Wed Sep* 13 12:14:43 *AEST* 2017 by *u5482878*