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

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

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

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

Invariants (formulas true in every reachable state.)

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

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

$StateOK \triangleq$ TRUE

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

Initialization

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

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

Helper Actions

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

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

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

Program Execution Actions

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

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

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

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

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

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

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

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

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

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

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

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

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

Kill(dead) ≜
  ∧ killed' = killed ∪ {dead}
  ∧ adoptSet' = adoptSet ∪ {m ∈ C!Adopter :
                              ∧ m.child ≠ C!NotID
                              ∧ m.adopter ≠ C!NotID
                              ∧ m.here ≠ dead
                              ∧ m.here = fstates[m.adopter].here
                              ∧ m.child ∈ fmasters[m.adopter].children
                              ∧ fbackups[m.child].isAdopted = FALSE
                              ∧ fstates[m.child].here = dead
                              ∧ m.adopter = fstates[m.child].eroot}
  ∧ IF adoptSet' = {}
    THEN ∧ mastersStatus' = [mastersStatus EXCEPT ![PROG_HOME].status = "convertDead",
                                                  ![PROG_HOME].lastKilled = dead]
         ∧ 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
                     ∧ msgs' = msgs
                     ∧ waitForMsgs' = waitForMsgs
                ELSE ∧ Kill(PROG[nested].dst)
  ∧ UNCHANGED ⟨fstates, pstate, seq, pendingAct, fmasters, fbackups,
               runningThrds, blockedThrds⟩
```

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

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

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

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

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

$$\begin{array}{rl}
& \quad dst \mapsto backupPlace, \\
& \quad target \mapsto asyncDst, \\
& \quad fid \mapsto root, \\
& \quad isAdopter \mapsto isAdopter, \\
& \quad adoptedFID \mapsto adoptedFID, \\
& \quad type \mapsto \text{“backupTransit”}]) \\
& \wedge thrds' = thrds \\
& \wedge blockedThrds' = blockedThrds \\
& \wedge runningThrds' = runningThrds \\
& \wedge waitForMsgs' = (waitForMsgs \setminus \{masterWFM\}) \cup \{backupWFM\} \\
& \wedge C!IncrMSEQ(1) \\
\text{ELSE IF} & \ success \wedge rootPlace \notin killed \ \text{ignore the } async\text{, go to the next step} \\
\text{THEN} & \wedge C!RecvMsg(msg) \\
& \wedge thrds' = [thrds \ \text{EXCEPT} \ ![here][tid].status = \text{“running”}, \\
& \qquad ![here][tid].stack = \\
& \qquad\quad \langle[\ b \mapsto top.b, \\
& \qquad\quad\ \ i \mapsto curStmt, \\
& \qquad\quad\ \ fid \mapsto fid] \\
& \qquad\quad \rangle \circ tail] \\
& \wedge blockedThrds' = blockedThrds \setminus \{[here \mapsto here,\ tid \mapsto tid]\} \\
& \wedge runningThrds' = runningThrds \cup \{[here \mapsto here,\ tid \mapsto tid]\} \\
& \wedge waitForMsgs' = waitForMsgs \setminus \{masterWFM\} \\
& \wedge C!IncrMSEQ(1) \\
\text{ELSE} & \wedge C!ReplaceMsg(msg, [\ 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 asyncDst, \\
& \qquad\qquad\qquad\qquad fid \mapsto root, \\
& \qquad\qquad\qquad\qquad type \mapsto \text{“backupGetAdopter”}, \\
& \qquad\qquad\qquad\qquad actionType \mapsto \text{“transit”}, \\
& \qquad\qquad\qquad\qquad aid \mapsto C!NotActivity.aid, \\
& \qquad\qquad\qquad\qquad finishEnd \mapsto \text{FALSE}]) \\
& \wedge thrds' = thrds \\
& \wedge blockedThrds' = blockedThrds \\
& \wedge runningThrds' = runningThrds \\
& \wedge waitForMsgs' = waitForMsgs \setminus \{masterWFM\} \\
& \qquad \text{we don't expect the backup to die} \\
& \qquad \text{that is why we don't add} \\
& \qquad \textit{backupGetAdopterDone} \text{ in } \textit{waitForMsgs} \\
& \wedge C!IncrMSEQ(1) \\
\wedge \text{UNCHANGED} & \langle convertSet, adoptSet, mastersStatus, fstates, pstate, killed, pendingAct, \\
& fmasters, fbackups\rangle
\end{array}$$

$$
\begin{array}{l}
MasterLiveDone \triangleq \\
\quad \land\ pstate = \text{“running”} \\
\quad \land\ pendingAct \neq \{\} \\
\quad \land\ msgs \neq \{\} \\
\quad \land\ \text{LET}\ msg \triangleq C!FindMSG(\text{“masterLiveDone”}) \\
\qquad \text{IN} \quad \land\ msg \neq C!NotMessage \\
\qquad\qquad \land\ \text{LET}\ here \triangleq msg.dst \\
\qquad\qquad\qquad actId \triangleq msg.aid \\
\qquad\qquad\qquad activity \triangleq FindPendingActivity(actId) \\
\qquad\qquad\qquad root \triangleq msg.fid \\
\qquad\qquad\qquad submit \triangleq msg.submit \\
\qquad\qquad\qquad success \triangleq msg.success \\
\qquad\qquad\qquad rootPlace \triangleq C!GetFinishHome(root) \\
\qquad\qquad\qquad isAdopter \triangleq msg.isAdopter \\
\qquad\qquad\qquad adoptedFID \triangleq msg.adoptedFID \\
\qquad\qquad\qquad backupPlace \triangleq msg.backupPlace \\
\qquad\qquad\qquad source \triangleq msg.source \\
\qquad\qquad\qquad target \triangleq msg.target \\
\qquad\qquad\qquad masterWFM \triangleq [src \mapsto rootPlace, \\
\qquad\qquad\qquad\qquad\qquad\qquad dst \mapsto here, \\
\qquad\qquad\qquad\qquad\qquad\qquad fid \mapsto root, \\
\qquad\qquad\qquad\qquad\qquad\qquad aid \mapsto actId, \\
\qquad\qquad\qquad\qquad\qquad\qquad source \mapsto source, \\
\qquad\qquad\qquad\qquad\qquad\qquad target \mapsto target, \\
\qquad\qquad\qquad\qquad\qquad\qquad type \mapsto \text{“masterLiveDone”}\ ] \\
\qquad\qquad\qquad backupWFM \triangleq [\ src \mapsto backupPlace, \\
\qquad\qquad\qquad\qquad\qquad\qquad dst \mapsto here, \\
\qquad\qquad\qquad\qquad\qquad\qquad fid \mapsto root, \\
\qquad\qquad\qquad\qquad\qquad\qquad aid \mapsto actId, \\
\qquad\qquad\qquad\qquad\qquad\qquad source \mapsto source, \\
\qquad\qquad\qquad\qquad\qquad\qquad target \mapsto here, \\
\qquad\qquad\qquad\qquad\qquad\qquad isAdopter \mapsto isAdopter, \\
\qquad\qquad\qquad\qquad\qquad\qquad adoptedFID \mapsto adoptedFID, \\
\qquad\qquad\qquad\qquad\qquad\qquad type \mapsto \text{“backupLiveDone”}\ ] \\
\qquad\qquad \text{IN} \quad \land\ SetActionNameAndDepth(\langle \text{“MasterLiveDone”}, here \rangle) \\
\qquad\qquad\qquad \land\ activity \neq C!NotActivity \\
\qquad\qquad\qquad \land\ fstates[activity.fid].here = here \\
\qquad\qquad\qquad \land\ \text{IF}\ success \land submit \land rootPlace \notin killed \\
\qquad\qquad\qquad\qquad \text{THEN} \ \land\ C!ReplaceMsg(msg, [\ mid \mapsto seq.mseq, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad src \mapsto here, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad dst \mapsto backupPlace, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad source \mapsto source, \\
\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 aid \mapsto actId,
\end{array}
$$

```
                                                    type  ↦ "backupLive",
                                                isAdopter ↦ isAdopter,
                                                adoptedFID ↦ adoptedFID])
                        ∧ waitForMsgs′ = (waitForMsgs \ {masterWFM}) ∪ {backupWFM}
                        ∧ C!IncrMSEQ(1)
                        ∧ pendingAct′ = pendingAct
                ELSE IF success ∧ rootPlace ∉ killed
                THEN    ∧ C!RecvMsg(msg)
                        ∧ pendingAct′ = pendingAct \ {activity}
                        ∧ seq′ = seq
                        ∧ waitForMsgs′ = waitForMsgs \ {masterWFM}
                ELSE    ∧ C!ReplaceMsg(msg, [  mid ↦ seq.mseq,
                                               src  ↦ here,
                                               dst  ↦ C!GetBackup(rootPlace),
                                            source ↦ source,
                                            target ↦ here,
                                               fid  ↦ root,
                                              type  ↦ "backupGetAdopter",
                                               aid  ↦ actId,
                                          finishEnd ↦ FALSE,
                                          actionType ↦ "live"])
                        ∧ waitForMsgs′ = waitForMsgs \ {masterWFM}
                            we don't expect backup to die
                            so we don't add
                            backupGetAdopterDone in waitForMsgs
                        ∧ C!IncrMSEQ(1)
                        ∧ pendingAct′ = pendingAct
  ∧ UNCHANGED ⟨convertSet, adoptSet, mastersStatus, fstates, pstate,
               thrds, killed,  fmasters, fbackups, blockedThrds, runningThrds⟩

MasterCompletedDone ≜
  ∧ pstate = "running"
  ∧ msgs ≠ {}
  ∧ LET msg ≜ C!FindMSG("masterCompletedDone")
    IN  ∧ msg ≠ C!NotMessage
        ∧ LET here ≜ msg.dst
              root ≜ msg.fid
              success ≜ msg.success
              rootPlace ≜ C!GetFinishHome(root)
              isAdopter ≜ msg.isAdopter
              backupPlace ≜ msg.backupPlace
              finishEnd ≜ msg.finishEnd
              masterWFM ≜ [ src  ↦ rootPlace,
                            dst  ↦ here,
                            target ↦ here,
```

$$
\begin{array}{rl}
 & \quad fid \mapsto root, \\
 & \quad isAdopter \mapsto isAdopter, \\
 & \quad type \mapsto \text{“masterCompletedDone”} \ ] \\
backupWFM \triangleq & [\quad src \mapsto backupPlace, \\
 & \quad dst \mapsto here, \\
 & \quad fid \mapsto root, \\
 & \quad target \mapsto here, \\
 & \quad isAdopter \mapsto isAdopter, \\
 & \quad type \mapsto \text{“backupCompletedDone”} \ ]
\end{array}
$$

IN $\land\ SetActionNameAndDepth(\langle \text{“MasterCompletedDone”}, here \rangle)$

$\land$ IF $success \land rootPlace \notin killed$

THEN $\land\ C!ReplaceMsg(msg, [mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto backupPlace,$
$target \mapsto here,$
$fid \mapsto root,$
$type \mapsto \text{“backupCompleted”},$
$finishEnd \mapsto finishEnd,$
$isAdopter \mapsto isAdopter])$

$\land$ IF $finishEnd$ THEN $waitForMsgs' = (waitForMsgs \setminus \{masterWFM\})$
ELSE $waitForMsgs' = (waitForMsgs \setminus \{masterWFM\}) \cup \{backupWFM\}$

$\land\ C!IncrMSEQ(1)$

ELSE $\land\ C!ReplaceMsg(msg, [\ mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto C!GetBackup(rootPlace),$
$source \mapsto C!NotPlace,$
$target \mapsto here,$
$fid \mapsto root,$
$type \mapsto \text{“backupGetAdopter”},$
$aid \mapsto C!NotActivity.aid,$
$finishEnd \mapsto \text{FALSE},$
$actionType \mapsto \text{“completed”}])$

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

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 $\land\ C!ReplaceMsg(msg, [mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto adoptedRootPlace,$
$target \mapsto msg.target,$
$fid \mapsto adoptedRoot,$
$finishEnd \mapsto msg.finishEnd,$

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{type} \mapsto \text{"adopterCompleted"}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{adoptedFID} \mapsto \mathit{adoptedFID}]) \\
\qquad\qquad\qquad\qquad \land\ C!\mathit{IncrMSEQ}(1) \\
\qquad\qquad\qquad \text{ELSE}\ \ \text{FALSE} \\
\land\ \text{UNCHANGED}\ \langle \mathit{fstates}, \mathit{pstate}, \mathit{thrds}, \mathit{killed}, \mathit{pendingAct}, \mathit{fmasters}, \mathit{fbackups}, \mathit{waitForMsgs}, \\
\qquad \mathit{mastersStatus}, \mathit{adoptSet}, \mathit{convertSet}, \mathit{blockedThrds}, \mathit{runningThrds} \rangle
\end{array}
$$

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

Terminated finish unblocks its thread

$$
\begin{array}{l}
\mathit{UnblockTerminateAsync} \triangleq \\
\quad \land\ \mathit{pstate} = \text{"running"} \\
\quad \land\ \text{LET}\ \mathit{pthrd} \triangleq \mathit{FindBlockedThreadAsyncTerm} \\
\qquad \text{IN}\ \ \land\ \mathit{pthrd} \neq C!\mathit{NotPlaceThread} \\
\qquad\qquad \land\ \text{LET}\ \mathit{here} \triangleq \mathit{pthrd}.\mathit{here} \\
\qquad\qquad\qquad \mathit{tid} \triangleq \mathit{pthrd}.\mathit{tid} \\
\qquad\qquad\qquad \mathit{msg} \triangleq C!\mathit{FindIncomingMSG}(\mathit{here}, \text{"backupCompletedDone"}) \\
\qquad\qquad\qquad \mathit{success} \triangleq \mathit{msg}.\mathit{success} \\
\qquad\qquad\qquad \mathit{top} \triangleq \mathit{Head}(\mathit{thrds}[\mathit{here}][\mathit{tid}].\mathit{stack}) \\
\qquad\qquad\qquad \mathit{blk} \triangleq \mathit{top}.b \\
\qquad\qquad\qquad \mathit{fid} \triangleq \mathit{top}.\mathit{fid} \\
\qquad\qquad\qquad \mathit{root} \triangleq \mathit{msg}.\mathit{fid} \\
\qquad\qquad\qquad \mathit{rootPlace} \triangleq C!\mathit{GetFinishHome}(\mathit{root}) \\
\qquad\qquad \text{IN}\ \ \land\ \mathit{SetActionNameAndDepth}(\langle \text{"UnblockTerminateAsync"}, \mathit{here}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{"success"}, \mathit{success} \rangle) \\
\qquad\qquad\qquad \land\ \mathit{waitForMsgs}' = \mathit{waitForMsgs} \setminus \{[\mathit{src} \mapsto \mathit{rootPlace}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{dst} \mapsto \mathit{here}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{target} \mapsto \mathit{here}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{fid} \mapsto \mathit{root}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{type} \mapsto \text{"backupCompletedDone"}\ ]\} \\
\qquad\qquad\qquad \land\ \text{IF}\ \ \mathit{Len}(\mathit{thrds}[\mathit{here}][\mathit{tid}].\mathit{stack}) = 1
\end{array}
$$

```
                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
```

$\land$ LET $submit \triangleq src \notin killed \land target \notin killed$
IN $\land$ 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 $fmasters' = fmasters$
$\land$ IF $src \in killed$
THEN $\land\ C!RecvMsg(msg)$
$\land\ seq' = seq$
ELSE $\land\ C!ReplaceMsg(msg,$ [$mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto src,$
$target \mapsto target,$
$fid \mapsto fid,$
$type \mapsto$ "masterTransitDone",
$submit \mapsto submit,$
$success \mapsto$ TRUE,
$isAdopter \mapsto$ FALSE,
$adoptedFID \mapsto C!NotID,$
$backupPlace \mapsto backupPlace$])
$\land\ C!IncrMSEQ(1)$
$\land$ UNCHANGED $\langle waitForMsgs, convertSet, adoptSet, mastersStatus, fstates, pstate,$
$thrds, killed, pendingAct, fbackups,$
$blockedThrds, runningThrds\rangle$

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

$\land$ LET $submit \triangleq source \notin killed \land target \notin killed$
IN $\land$ IF $submit$
THEN $\land fmasters[fid].transit[source][target] > 0$
$\land fmasters' = [fmasters$ EXCEPT $![fid].transit[source][target] = @ - 1,$
$![fid].live[target] = @ + 1]$
ELSE $\land fmasters' = fmasters$
$\land$ IF $target \in killed$
THEN $\land C!RecvMsg(msg)$
$\land seq' = seq$
ELSE $\land C!ReplaceMsg(msg, [\ mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto target,$
$source \mapsto source,$
$target \mapsto target,$
$fid \mapsto fid,$
$aid \mapsto msg.aid,$
$type \mapsto$ "masterLiveDone",
$submit \mapsto submit,$
$success \mapsto$ TRUE,
$isAdopter \mapsto$ FALSE,
$adoptedFID \mapsto C!NotID,$
$backupPlace \mapsto backupPlace])$
$\land C!IncrMSEQ(1)$
$\land$ UNCHANGED $\langle convertSet, adoptSet, mastersStatus, fstates, pstate,$
$thrds, waitForMsgs, killed, pendingAct, fbackups,$
$blockedThrds, runningThrds\rangle$

$MasterCompleted \triangleq$
$\land pstate =$ "running"
$\land msgs \neq \{\}$
$\land$ LET $msg \triangleq C!FindMSG($"masterCompleted"$)$
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 $\land SetActionNameAndDepth(\langle$"MasterCompleted", $here\rangle)$
$\land backupPlace \neq C!NotPlace$
$\land fid \neq C!NotID$
$\land fstates[fid].here = here$
$\land target = src$
$\land mastersStatus[here].status =$ "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

$AdopterTransit \triangleq$
$\land\ pstate = \text{“running”}$
$\land\ msgs \neq \{\}$
$\land$ LET $msg \triangleq C!FindMSG(\text{“adopterTransit”})$
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 $\land\ SetActionNameAndDepth(\langle \text{“AdopterTransit”}, here\rangle)$
$\land\ mastersStatus[here].status = \text{“running”}$
$\land\ fid \neq C!NotID$
$\land\ fstates[fid].here = here$
$\land$ 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 $\land\ fmasters' = fmasters$
$\land\ C!ReplaceMsg(msg,\ [\ mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto src,$
$target \mapsto target,$
$fid \mapsto fid,$
$type \mapsto \text{“masterTransitDone”},$
$submit \mapsto submit,$
$success \mapsto$ TRUE,
$backupPlace \mapsto backupPlace,$
$isAdopter \mapsto$ TRUE,
$adoptedFID \mapsto adoptedFID\ ])$
$\land\ C!IncrMSEQ(1)$
$\land$ UNCHANGED $\langle convertSet,\ adoptSet,\ mastersStatus,\ fstates,\ pstate,$
$thrds,\ killed,\ pendingAct,\ fbackups,\ waitForMsgs,$
$blockedThrds,\ runningThrds\rangle$

$AdopterLive \triangleq$
$\land\ pstate = \text{“running”}$
$\land\ msgs \neq \{\}$

$\land$ LET $msg \triangleq C!FindMSG(\text{“adopterLive”})$
IN $\land\ msg \neq C!NotMessage$
$\land$ LET $here \triangleq msg.dst$
$fid \triangleq msg.fid$
$source \triangleq msg.source$
$target \triangleq msg.target$
$backupPlace \triangleq C!GetBackup(here)$
$adoptedFID \triangleq msg.adoptedFID$
IN $\land\ SetActionNameAndDepth(\langle \text{“AdopterLive”}, here \rangle)$
$\land\ fid \neq C!NotID$
$\land\ backupPlace \neq C!NotPlace$
$\land\ fstates[fid].here = here$
$\land\ mastersStatus[here].status = \text{“running”}$
$\land\ target = msg.src$
$\land$ LET $submit \triangleq source \notin killed \ \land \ target \notin killed$
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 $fmasters' = fmasters$
$\land\ C!ReplaceMsg(msg,\ [\ mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto target,$
$source \mapsto source,$
$target \mapsto target,$
$fid \mapsto fid,$
$aid \mapsto msg.aid,$
$type \mapsto \text{“masterLiveDone”},$
$submit \mapsto submit,$
$success \mapsto$ TRUE,
$isAdopter \mapsto$ TRUE,
$adoptedFID \mapsto adoptedFID,$
$backupPlace \mapsto backupPlace])$
$\land\ C!IncrMSEQ(1)$
$\land$ UNCHANGED $\langle convertSet, adoptSet, mastersStatus, fstates, pstate, waitForMsgs,$
$thrds, waitForMsgs, killed, pendingAct, fbackups,$
$blockedThrds, runningThrds\rangle$

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

$\land\ C!IncrMSEQ(1)$

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

Finish backup replica actions

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

$BackupTransit\ \triangleq$
$\land\ pstate = \text{“running”}$
$\land\ msgs \neq \{\}$
$\land$ 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 $\land SetActionNameAndDepth(\langle$"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 $\land C!ReplaceMsg(msg, [\ mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto src,$
$target \mapsto target,$
$fid \mapsto fid,$
$type \mapsto$ "masterTransitDone",
$isAdopter \mapsto$ FALSE,
$adoptedFID \mapsto C!NotID,$
$backupPlace \mapsto C!NotPlace,$
$submit \mapsto$ FALSE,
$success \mapsto$ FALSE$]\ )$
$\land C!IncrMSEQ(1)$
ELSE IF $src \in killed$
THEN $\land C!RecvMsg(msg)$
$\land seq' = seq$
ELSE $\land C!ReplaceMsg(msg, [\ mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto src,$
$target \mapsto target,$
$fid \mapsto fid,$
$type \mapsto$ "backupTransitDone",
$success \mapsto$ TRUE,
$isAdopter \mapsto isAdopter,$
$adoptedFID \mapsto adoptedFID])$
$\land C!IncrMSEQ(1)$

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

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

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

$\land$ IF $fbackups[fid].isAdopted$ Change to the path of *adopterLive*
THEN $\land\ C!ReplaceMsg(msg, [\ mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto src,$
$source \mapsto source,$
$target \mapsto target,$
$fid \mapsto fid,$
$aid \mapsto msg.aid,$
$type \mapsto \text{“masterLiveDone”},$
$submit \mapsto$ FALSE,
$success \mapsto$ FALSE,
$isAdopter \mapsto$ FALSE,
$adoptedFID \mapsto C!NotID,$
$backupPlace \mapsto C!NotPlace])$
$\land\ C!IncrMSEQ(1)$
ELSE IF $src \in killed$
THEN $\land\ C!RecvMsg(msg)$
$\land\ seq' = seq$

```
                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])
```

$\land\ C!IncrMSEQ(1)$
ELSE IF $src \in killed \lor finishEnd$
THEN $\land\ C!RecvMsg(msg)$
$\land\ seq' = seq$
ELSE $\land\ C!ReplaceMsg(msg,\ [\ mid \mapsto seq.mseq,$
$src \mapsto here,$
$dst \mapsto src,$
$target \mapsto target,$
$fid \mapsto fid,$
$isAdopter \mapsto isAdopter,$
$type \mapsto$ "backupCompletedDone",
$success \mapsto$ TRUE$])$
$\land\ C!IncrMSEQ(1)$
$\land$ UNCHANGED $\langle convertSet,\ adoptSet,\ mastersStatus,\ fstates,\ pstate,$
$thrds,\ killed,\ pendingAct,\ \ fmasters,\ waitForMsgs,$
$blockedThrds,\ runningThrds\rangle$

Finish adoption actions for recovery

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

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

$\land\ convertSet' = \ convertSet \cup \{t \in C!ConvTask :$
$\land\ t.pl \neq C!NotPlace$
$\land\ t.pl \notin killed$
$\land\ t.fid = adopter$
$\land\ t.here = here\}$
$\land$ IF $\exists\, m \in adoptSet' : m.here = here$
THEN $\land\ mastersStatus' = mastersStatus$
ELSE $\land\ mastersStatus' = [mastersStatus$ EXCEPT $![here].status =$ "convertDead"$]$
$\land$ UNCHANGED $\langle fstates, msgs, pstate, seq, thrds, killed, pendingAct, \ waitForMsgs,$
$blockedThrds, runningThrds\rangle$

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

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

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

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

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

$Next \triangleq$

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

Asserting fairness properties to all actions

$Liveness \triangleq$

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

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

Specification

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

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

* Modification History
* Last modified *Mon Dec* 11 21:17:12 *AEDT* 2017 by *u*5482878
* Last modified Sun *Dec* 10 18:15:04 *AEDT* 2017 by *shamouda*
* Created *Wed Sep* 13 12:14:43 *AEST* 2017 by *u*5482878