MODULE *ExecutorDistFinishOptimistic*

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. Unlike the pessimistic finish protocol proposed in $PPoPP14$, we are specifying a new optimistic protocol that reduces communication during normal execution, at the expense of more complex recovery

Assumptions:

- A root finish will have at most one remote finish at any other place
- Remote finish objects must be cleaned after root finish is released

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,	All finish states (root and remote)
fmasters,	Root finishes master states
fbackups,	Root finishes backup 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
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
convFromDead,	Recovery variable: set of finishes having transit tasks from a dead place
convToDead,	Recovery variable: set of finishes having transit

```
                              tasks to a dead place
    actionName,               Debugging variable: the current action name
    depth                     Debugging variable: the current depth
```

$Vars \triangleq \langle fstates, msgs, pstate, seq, thrds,$
$\quad killed, fmasters, fbackups, waitForMsgs,$
$\quad mastersStatus, convFromDead, convToDead,$
$\quad blockedThrds, runningThrds, actionName, depth\rangle$

Predicate to hide the finish implementation

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

$C \triangleq \text{INSTANCE } Commons$

```
GetRootFinishId(fid) ≜
  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 ≜
  ∧ $fstates \in [C!IDRange \rightarrow C!FinishState]$
  ∧ $thrds \in [PLACE \rightarrow [C!ThreadID \rightarrow C!Thread]]$
  ∧ $msgs \subseteq C!Messages$
  ∧ $pstate \in \{\text{"running"}, \text{"terminated"}\}$
  ∧ $PROG \in [C!BlockID \rightarrow C!Block]$
  ∧ $PROG\_HOME \in PLACE$
  ∧ $seq \in C!Sequences$
  ∧ $killed \subseteq PLACE$
  ∧ $fmasters \in [C!IDRange \rightarrow C!MasterFinish]$
  ∧ $fbackups \in [C!IDRange \rightarrow C!BackupFinish]$
  ∧ $BACKUP \in [PLACE \rightarrow PLACE]$
  ∧ $mastersStatus \in [PLACE \rightarrow C!MasterStatus]$
  ∧ $convFromDead \subseteq C!ConvTask$
  ∧ $convToDead \subseteq C!ConvTask$
  ∧ $runningThrds \subseteq C!PlaceThread$
  ∧ $blockedThrds \subseteq C!PlaceThread$
  ∧ $depth \in 0 \,..\, DEPTH + 1$
```

$StateOK \triangleq \text{TRUE}$

$MustTerminate \triangleq$
$\quad \Diamond(pstate = \text{"terminated"})$

Initialization

```
$Init \triangleq$
    $\land\ actionName = \langle \text{“Init”}, PROG\_HOME \rangle$
    $\land\ depth = 0$
    $\land\ fstates = [r \in C!IDRange \mapsto$
                   $[id \mapsto C!NotID, status \mapsto \text{“unused”}, type \mapsto \text{“NA”},$
                    $count \mapsto 0, here \mapsto C!NotPlace,$
                    $parent \mapsto C!NotID, root \mapsto C!NotID, isGlobal \mapsto \text{FALSE},$
                    $eroot \mapsto C!NotID, deny \mapsto \{\}, newMaster \mapsto C!NotPlace,$
                    $newBackup \mapsto C!NotPlace, src \mapsto C!NotPlace,$
                    $received \mapsto [p \in PLACE \mapsto 0]]]$
    $\land\ fmasters = [r \in C!IDRange \mapsto$
                          $[\ id \mapsto C!NotID,$
                            $src \mapsto C!NotPlace,$
                            $home \mapsto C!NotPlace,$
                            $numActive \mapsto 0,$
                            $transit \mapsto [p \in PLACE \mapsto [q \in PLACE \mapsto 0]],$
                            $adoptedChildren \mapsto \{\},$
                            $newBackup \mapsto C!NotPlace,$
                            $isAdopted \mapsto \text{FALSE},$
                            $isReleased \mapsto \text{FALSE},$
                            $adopterPlace \mapsto C!NotPlace,$
                            $\_src \mapsto C!NotPlace,$
                            $\_home \mapsto C!NotPlace,$
                            $\_numActive \mapsto 0,$
                            $\_transit \mapsto [p \in PLACE \mapsto [q \in PLACE \mapsto 0]],$
                            $\_adoptedChildren \mapsto \{\},$
                            $\_newBackup \mapsto C!NotPlace,$
                            $\_isAdopted \mapsto \text{FALSE},$
                            $\_isReleased \mapsto \text{FALSE},$
                            $\_adopterPlace \mapsto C!NotPlace$
                          $]]$
    $\land\ fbackups = [r \in C!IDRange \mapsto$
                          $[\ id \mapsto C!NotID,$
                            $src \mapsto C!NotPlace,$
                            $home \mapsto C!NotPlace,$
                            $numActive \mapsto 0,$
                            $transit \mapsto [p \in PLACE \mapsto [q \in PLACE \mapsto 0]],$
                            $adoptedChildren \mapsto \{\},$
                            $newMaster \mapsto C!NotPlace,$
                            $isAdopted \mapsto \text{FALSE},$
                            $isReleased \mapsto \text{FALSE},$
                            $adopterPlace \mapsto C!NotPlace,$
                            $\_src \mapsto C!NotPlace,$
                            $\_home \mapsto C!NotPlace,$
                    $\_numActive \mapsto 0,$
                    $\_transit \mapsto [p \in PLACE \mapsto [q \in PLACE \mapsto 0]],$
                    $\_adoptedChildren \mapsto \{\},$
                    $\_newMaster \mapsto C!NotPlace,$
                    $\_isAdopted \mapsto \text{FALSE},$
                    $\_isReleased \mapsto \text{FALSE},$
                    $\_adopterPlace \mapsto C!NotPlace$
                    $]]$
$\land pstate = \text{"running"}$
$\land mastersStatus = [p \in PLACE \mapsto [\ status \mapsto \text{"running"},$
                                            $lastKilled \mapsto C!NotPlace]]$
$\land msgs = \{\}$
$\land seq = [aseq \mapsto 1, fseq \mapsto C!FIRST\_ID, mseq \mapsto 1]$
$\land thrds = [p \in PLACE \mapsto$   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 $C!I\_START,$
                                 $fid \mapsto C!NoParent,$
                                 $src \mapsto PROG\_HOME]\rangle]$
           ELSE $[tid \mapsto t, status \mapsto \text{"idle"},$
                 $blockingType \mapsto \text{"NA"},$
                 $stack \mapsto \langle\rangle]]]$
$\land runningThrds = \{[here \mapsto PROG\_HOME, tid \mapsto 0]\}$
$\land blockedThrds = \{\}$
$\land killed = \{\}$
$\land waitForMsgs = \{\}$
$\land convFromDead = \{\}$
$\land convToDead = \{\}$
```

Helper Actions

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

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

Program Execution Actions

```
FindRunningThreadForStartFinish $\triangleq$
  LET $tset \triangleq \{t \in runningThrds$ :
                  $\land t.here \notin killed$
                  $\land thrds[t.here][t.tid].status =$ "running"
                  $\land$ LET $top \triangleq Head(thrds[t.here][t.tid].stack)$
                          $blk \triangleq top.b$
                          $lstStmt \triangleq top.i$
                     IN   $\land PROG[blk].type =$ "finish"
                          $\land lstStmt = C!I\_PRE\_FIN\_ALLOC\}$
  IN   IF $tset = \{\}$ THEN $C!NotPlaceThread$
         ELSE  CHOOSE $x \in tset$ : TRUE
```

Running thread processing the beginning of a finish block

```
StartFinish $\triangleq$
   $\land pstate =$ "running"
   $\land$ LET $pthrd \triangleq FindRunningThreadForStartFinish$
      IN   $\land pthrd \neq C!NotPlaceThread$
           $\land$ LET $here \triangleq pthrd.here$
                   $tid \triangleq pthrd.tid$
                   $top \triangleq Head(thrds[here][tid].stack)$
                   $tail \triangleq Tail(thrds[here][tid].stack)$
                   $lstStmt \triangleq top.i$
                   $curStmt \triangleq top.i + 1$
                   $blk \triangleq top.b$
                   $fid \triangleq top.fid$
                   $newFid \triangleq seq.fseq$
                   $encRoot \triangleq C!GetEnclosingRoot(fid, newFid)$
              IN   $\land SetActionNameAndDepth(\langle$"StartFinish", $here\rangle)$
                   $\land Finish(seq.fseq)!Alloc(C!ROOT\_FINISH, here, fid, newFid, top.src)$
                   $\land C!IncrFSEQ$
                   $\land thrds' = [thrds$ EXCEPT $![here][tid].stack =$
                                              $\langle[b \mapsto top.b,$
                                                 $i \mapsto curStmt,$
                                                 $fid \mapsto seq.fseq,$
                                                 $src \mapsto top.src]$
                                              $\rangle \circ tail]$
                   $\land$ IF $seq.fseq = C!FIRST\_ID$
                        THEN $\land fmasters' = fmasters$  will be initialized in transit
                             $\land fbackups' = fbackups$
```

```
                    ELSE  $\land$ $fmasters' = [fmasters$ EXCEPT $![encRoot].children =$
                                                       $@ \cup \{newFid\}]$
                          $\land$ $fbackups' = [fbackups$ EXCEPT $![encRoot].children =$
                                                       $@ \cup \{newFid\}]$
  $\land$ UNCHANGED $\langle convFromDead, convToDead, mastersStatus, pstate, killed,$
                    $msgs, waitForMsgs, runningThrds, blockedThrds \rangle$
```

```
$FindRunningThreadForScheduleNestedFinish \triangleq$
  LET $tset \triangleq \{t \in runningThrds :$
              $\land$ $t.here \notin killed$
              $\land$ $thrds[t.here][t.tid].status =$ "running"
              $\land$ LET $top \triangleq Head(thrds[t.here][t.tid].stack)$
                      $blk \triangleq top.b$
                      $curStmt \triangleq top.i + 1$
                      $nested \triangleq PROG[blk].stmts[curStmt]$
                 IN   $\land$ $PROG[blk].type \notin \{$"expr", "kill"$\}$
                      $\land$ $curStmt \geq 0$
                      $\land$ $curStmt \leq PROG[blk].mxstmt$
                      $\land$ $PROG[nested].type =$ "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

```
$ScheduleNestedFinish \triangleq$
  $\land$ $pstate =$ "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   $\land$ $SetActionNameAndDepth(\langle$"ScheduleNestedFinish", $here\rangle)$
                  $\land$ $thrds' = [thrds$ EXCEPT $![here][tid].stack =$
                                              $\langle[\ \ b \mapsto nested,$
                                                   $i \mapsto C!I\_START,$
                                                 $fid \mapsto newFid,$
```

```
                                                              $src \mapsto top.src]$,
                                                           $[\ \ b \mapsto top.b$,
                                                              $i \mapsto curStmt$,
                                                              $fid \mapsto fid$,
                                                              $src \mapsto top.src]$
                                                           $\rangle \circ tail]$
                        $\land\ Finish(seq.fseq)!Alloc(C!ROOT\_FINISH,\ here,\ fid,\ newFid,\ top.src)$
                        $\land\ C!IncrFSEQ$
  $\land$ UNCHANGED $\langle convFromDead,\ convToDead,\ mastersStatus,\ msgs,\ pstate,\ waitForMsgs,$
                    $killed,\ runningThrds,\ blockedThrds,\ fmasters,\ fbackups\rangle$
```

```
$FindRunningThreadForSpawnLocalAsync \triangleq$
  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 $here \triangleq pthrd.here$
                 $tid \triangleq pthrd.tid$
                 $top \triangleq Head(thrds[here][tid].stack)$
                 $tail \triangleq Tail(thrds[here][tid].stack)$
                 $lstStmt \triangleq top.i$
                 $curStmt \triangleq top.i + 1$
                 $blk \triangleq top.b$
                 $fid \triangleq top.fid$
                 $nested \triangleq PROG[blk].stmts[curStmt]$
                 $idle \triangleq FindIdleThread(here)$
                 $act \triangleq [aid \mapsto seq.aseq,\ b \mapsto nested,\ fid \mapsto fid,\ src \mapsto top.src]$
```

```
                  $stkEntry \triangleq [b \mapsto act.b,\ i \mapsto C!I\_START,\ fid \mapsto act.fid,\ src \mapsto top.src]$
            IN    $\land$ $SetActionNameAndDepth(\langle$"SpawnLocalAsync"$,\ here\rangle)$
                  $\land$ IF $act.fid \neq C!NoParent$
                       THEN $Finish(act.fid)!NotifyLocalActivitySpawnAndCreation(here,\ act)$
                       ELSE $fstates' = fstates$
                  $\land$ $C!IncrASEQ$
                  $\land$ $thrds' = [thrds$ EXCEPT $![here][tid].stack =$
                                                  $\langle[\ \ b \mapsto top.b,$
                                                  $\ \ \ \ i \mapsto curStmt,$
                                                  $\ \ \ fid \mapsto fid,$
                                                  $\ \ \ src \mapsto top.src]$
                                                  $\rangle \circ tail,$
                                              $![here][idle.tid].stack = \langle stkEntry\rangle,$
                                              $![here][idle.tid].status =$ "running"$]$
                  $\land$ $runningThrds' = runningThrds \cup \{[here \mapsto here,\ tid \mapsto idle.tid]\}$
     $\land$ UNCHANGED $\langle convFromDead,\ convToDead,\ mastersStatus,\ msgs,\ pstate,\ killed,$
                  $fmasters,\ fbackups,\ waitForMsgs,\ blockedThrds\rangle$
```

```
$FindRunningThreadForSpawnRemoteAsync \triangleq$
  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 $\langle convFromDead, convToDead, mastersStatus, pstate, killed,$
                $fmasters, fbackups \rangle$
```

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

$Kill(dead) \triangleq$
  $\land killed' = killed \cup \{dead\}$
  $\land mastersStatus' = [p \in PLACE \mapsto$ IF $p \neq dead$
                                        THEN $[$    $status \mapsto$ "preConvert",
                                             $lastKilled \mapsto dead]$
                                        ELSE $mastersStatus[p]]$
  $\land$ LET $delMsgs \triangleq \{m \in msgs : m.dst = dead \}$        delete messages going to a dead place
          $wfm \triangleq \{m \in waitForMsgs : m.dst = dead\}$   delete $waitForMsgs$ to a dead place
      IN  $\land msgs' = msgs \setminus delMsgs$
          $\land waitForMsgs' = waitForMsgs \setminus wfm$
  $\land$ LET $mastersWObackups \triangleq \{id \in C!IDRange :$
```

```
                                    ∧ fmasters[id].id ≠ C!NotID
                                    ∧ fmasters[id].newBackup = C!NotPlace
                                    ∧ BACKUP[fstates[id].here] = dead
                                    ∧ fstates[id].type = "distroot"}
      backupsWOmasters ≜ {id ∈ C!IDRange :
                                    ∧ fbackups[id].id ≠ C!NotID
                                    ∧ fstates[id].here = dead
                                    ∧ fbackups[id].newMaster = C!NotPlace
                                    ∧ fstates[id].type = "distroot"}
IN    ∧ SetActionNameAndDepth(⟨"RunExprOrKill", mastersWObackups, backupsWOmasters ⟩)
      ∧ fmasters' = [r ∈  C!IDRange  ↦
                       IF r ∈ mastersWObackups
                        THEN [fmasters[r] EXCEPT !.newBackup = BACKUP[dead]]
                        ELSE  IF r ∈ backupsWOmasters
                        THEN [fmasters[r] EXCEPT !.src = fbackups[r].src,
                                                 !.home = fbackups[r].home,
                                                 !.numActive = fbackups[r].numActive,
                                                 !.transit = fbackups[r].transit,
                                                 !.adoptedChildren = fbackups[r].adoptedChildren,
                                                 !.newBackup = BACKUP[fstates[r].here],   fixme, do I nee
                                                 !.isAdopted   = fbackups[r].isAdopted,
                                                 !.adopterPlace = fbackups[r].adopterPlace,
                                                 !.isReleased = fbackups[r].isReleased,
                                                 !._src = fmasters[r].src,
                                                 !._home = fmasters[r].home,
                                                 !._numActive = fmasters[r].numActive,
                                                 !._transit = fmasters[r].transit,
                                                 !._adoptedChildren = fmasters[r].adoptedChildren,
                                                 !._newBackup = fmasters[r].newBackup,
                                                 !._isAdopted   = fmasters[r].isAdopted,
                                                 !._adopterPlace = fmasters[r].adopterPlace,
                                                 !._isReleased = fmasters[r].isReleased]
                        ELSE  fmasters[r]]
 ∧ fstates' = [ r ∈ C!IDRange ↦
          IF  r ∈ backupsWOmasters
           THEN  [ fstates[r] EXCEPT !.newMaster = PROG_HOME ]  ELSE  fstates[r] ]
     ∧ fbackups' = [r ∈  C!IDRange  ↦
                      IF r ∈ mastersWObackups
                       THEN [fbackups[r] EXCEPT !.src = fmasters[r].src,
                                                !.home = fmasters[r].home,
                                                !.numActive = fmasters[r].numActive,
                                                !.transit = fmasters[r].transit,
                                                !.adoptedChildren = fmasters[r].adoptedChildren,
                                                !.newMaster = fbackups[r].newMaster,
                                                !.isAdopted   = fmasters[r].isAdopted,
                                                $!.adopterPlace = fmasters[r].adopterPlace,$
                                                $!.isReleased = fmasters[r].isReleased,$
                                                $!.\_src = fbackups[r].src,$
                                                $!.\_home = fbackups[r].home,$
                                                $!.\_numActive = fbackups[r].numActive,$
                                                $!.\_transit = fbackups[r].transit,$
                                                $!.\_adoptedChildren = fbackups[r].adoptedChildren,$
                                                $!.\_newMaster = fbackups[r].newMaster,$
                                                $!.\_isAdopted = fbackups[r].isAdopted,$
                                                $!.\_adopterPlace = fbackups[r].adopterPlace,$
                                                $!.\_isReleased = fbackups[r].isReleased]$
                          ELSE IF $r \in backupsWOmasters$
                          THEN $[fbackups[r]$ EXCEPT $!.newMaster = PROG\_HOME]$
                          ELSE $fbackups[r]$ ]
```

Processing a nested expression in the currently running block

```
$RunExprOrKill \triangleq$
  $\land pstate =$ "running"
  $\land$ LET $pthrd \triangleq FindRunningThreadForRunExprOrKill$
     IN  $\land pthrd \neq C!NotPlaceThread$
         $\land$ LET $here \triangleq pthrd.here$
                 $tid \triangleq pthrd.tid$
                 $top \triangleq Head(thrds[here][tid].stack)$
                 $tail \triangleq Tail(thrds[here][tid].stack)$
                 $lstStmt \triangleq top.i$
                 $curStmt \triangleq top.i + 1$
                 $blk \triangleq top.b$
                 $fid \triangleq top.fid$
                 $nested \triangleq PROG[blk].stmts[curStmt]$
            IN   $\land thrds' = [thrds$ EXCEPT $![here][tid].stack =$
                                                        $\langle[\ b \mapsto top.b,$
                                                           $i \mapsto curStmt,$
                                                           $fid \mapsto fid,$
                                                           $src \mapsto top.src]$
                                                        $\rangle \circ tail]$
                 $\land$ IF $PROG[nested].type =$ "expr"
                      THEN $\land killed' = killed$
                           $\land PROG[nested].dst = here$
                           $\land mastersStatus' = mastersStatus$
                           $\land msgs' = msgs$
                           $\land waitForMsgs' = waitForMsgs$
                           $\land fmasters' = fmasters$
                           $\land fbackups' = fbackups$
                           $\land SetActionNameAndDepth(\langle$"RunExprOrKill"$, here, PROG[nested].type\rangle)$
                      ELSE $\land Kill(PROG[nested].dst)$
```

```
    ∧ UNCHANGED ⟨fstates, pstate, seq, runningThrds, blockedThrds,
                 convFromDead, convToDead⟩
```

```
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(top.src, 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 ⟨convFromDead, convToDead, mastersStatus, pstate, killed,
               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

```
$StopFinish \triangleq$
  $\land$ $pstate =$ "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"
                  $\land$ $PROG[top.b].mxstmt = top.i$
                  $\land$ $Finish(top.fid)!NotifyActivityTermination(top.src, \text{TRUE})$
                  $\land$ $thrds' = [thrds$ EXCEPT $![here][tid].status =$ "blocked",
                                                $![here][tid].blockingType =$ "FinishEnd"$]$
                  $\land$ $runningThrds' = runningThrds \setminus \{[here \mapsto here, tid \mapsto tid]\}$
                  $\land$ $blockedThrds' = blockedThrds \cup \{[here \mapsto here, tid \mapsto tid]\}$
  $\land$ UNCHANGED $\langle convFromDead, convToDead, mastersStatus, pstate, killed,$
                    $fmasters, fbackups \rangle$
```

```
$RecvAsync \triangleq$
  $\land$ $pstate =$ "running"
  $\land$ LET $msg \triangleq C!FindMSG($"async"$)$
     IN   $\land$ $msg \neq C!NotMessage$
          $\land$ LET $here \triangleq msg.dst$
                  $pid \triangleq msg.fid$
                  $fid \triangleq C!GetActiveFID(C!REMOTE\_FINISH, here, pid)$
                  $src \triangleq msg.src$
                  $blk \triangleq msg.b$
                  $newFID \triangleq$ IF $fid = C!NotID$ THEN $seq.fseq$ ELSE $fid$
                  $activity \triangleq [aid \mapsto seq.aseq, b \mapsto blk, fid \mapsto newFID, src \mapsto src]$
                  $denyList \triangleq$ IF $fid = C!NotID$ THEN $\{\}$ ELSE $fstates[fid]$
                  $accept \triangleq$ $\lor (fid \neq C!NotID \land src \notin denyList)$
                                    $\lor (fid = C!NotID \land src \notin killed)$
             IN   $\land$ $SetActionNameAndDepth(\langle$ "RecvAsync", $here$, "accept", $accept \rangle)$
                  $\land$ $pid \neq C!NotID$
                  $\land$ $src \neq C!NotPlace$
                  $\land$ IF $(fid \neq C!NotID \land src \notin denyList)$
                       THEN $Finish(activity.fid)!NotifyRemoteActivityCreation($
                                        $src, activity, msg, C!REMOTE\_FINISH,$
                                        $here, pid, pid, activity.src)$
                       ELSE IF $(fid = C!NotID \land src \notin killed)$
                       THEN $Finish(activity.fid)!AllocRemoteAndNotifyRemoteActivityCreation($
                                        $src, activity, msg, C!REMOTE\_FINISH,$
                                        $here, pid, pid, activity.src)$
```

```
                ELSE  $\land fstates' = fstates$
                      $\land C!RecvMsg(msg)$
            $\land$ IF $(\neg accept\ )$
                THEN  $\land thrds' = thrds$
                      $\land runningThrds' = runningThrds$
                ELSE  LET $idleThrd \triangleq FindIdleThread(here)$
                          $stkEntry \triangleq [b \mapsto activity.b,\ i \mapsto C!I\_START,$
                                          $fid \ \ \mapsto activity.fid,\ src \mapsto activity.src]$
                      IN   $\land thrds' = [thrds$ EXCEPT $![here][idleThrd.tid].stack = \langle stkEntry \rangle,$
                                                   $![here][idleThrd.tid].status =$ “running” ]
                           $\land runningThrds' = runningThrds \cup \{[here \mapsto here,\ tid \mapsto idleThrd.tid]\}$
            $\land C!IncrAll$
  $\land$ UNCHANGED $\langle convFromDead,\ convToDead,\ mastersStatus,\ pstate,$
                  $killed,\ fmasters,\ fbackups,\ blockedThrds,\ waitForMsgs\rangle$
```

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

$MasterTransitDone \triangleq$
  $\land pstate =$ “running”
  $\land msgs \neq \{\}$
  $\land$ LET $pthrd \triangleq FindBlockedThreadMasterTransitDone$
     IN    $\land pthrd \neq C!NotPlaceThread$
           $\land$ LET $here \triangleq pthrd.here$
                   $tid \ \triangleq pthrd.tid$
                   $msg \triangleq C!FindIncomingMSG(here,$ “masterTransitDone” $)$
                   $success \triangleq msg.success$
                   $submit \triangleq msg.submit$
                   $top \triangleq Head(thrds[here][tid].stack)$
                   $tail \triangleq Tail(thrds[here][tid].stack)$
                   $lstStmt \ \triangleq top.i$
                   $curStmt \triangleq top.i + 1$
                   $blk \triangleq top.b$
                   $root \triangleq msg.fid$
                   $fid \triangleq top.fid$
                   $rootPlace \triangleq C!GetFinishHome(root)$
                   $nested \triangleq PROG[blk].stmts[curStmt]$
                   $asyncDst \triangleq PROG[nested].dst$
```

```
          $backupPlace \triangleq msg.backupPlace$
          $finishSrc \triangleq msg.finishSrc$
          $masterWFM \triangleq [src \mapsto rootPlace,$
                              $dst \mapsto here,$
                              $fid \mapsto root,$
                              $target \mapsto asyncDst,$
                              $finishSrc \mapsto finishSrc,$
                              $taskFID \mapsto msg.taskFID,$
                              $type \mapsto$ "masterTransitDone" ]
          $backupWFM \triangleq [src \mapsto backupPlace,$
                              $dst \mapsto here,$
                              $fid \mapsto root,$
                              $target \mapsto asyncDst,$
                              $finishSrc \mapsto finishSrc,$
                              $type \mapsto$ "backupTransitDone" ]
      IN  $\land SetActionNameAndDepth(\langle$ "MasterTransitDone", $here,$
                                       "success", $success,$
                                       "submit", $submit\rangle)$
          $\land fid = msg.taskFID$
          Technically, we should check the condition $rootPlace \notin killed$
          if success is true. we should communicate with the backup normally.
          the backup then should reject the request and notify the requester
          that the master has changed, so that we redirect the call to the
          new master.
          $\land$ IF $success \land submit$
             THEN $\land C!ReplaceMsg(msg, [$ $mid \mapsto seq.mseq,$
                                              $src \mapsto here,$
                                              $dst \mapsto backupPlace,$
                                              $target \mapsto asyncDst,$
                                              $fid \mapsto root,$
                                              $finishSrc \mapsto finishSrc,$
                                              $knownMaster \mapsto msg.src,$
                                              $taskFID \mapsto fid,$
                                              $type \mapsto$ "backupTransit"])
                  $\land thrds' = thrds$
                  $\land blockedThrds' = blockedThrds$
                  $\land runningThrds' = runningThrds$
                  $\land waitForMsgs' = (waitForMsgs \setminus \{masterWFM\}) \cup \{backupWFM\}$
                  $\land C!IncrMSEQ(1)$
             ELSE IF $success$ ignore the $async$, go to the next step
             THEN $\land C!RecvMsg(msg)$
                  $\land thrds' = [thrds$ EXCEPT $![here][tid].status =$ "running",
                                             $![here][tid].stack =$
                                                  $\langle[\ b \mapsto top.b,$
                                                     $i \mapsto curStmt,$
```

```
                                                          fid ↦ fid,
                                                          src ↦ top.src]
                                                        ⟩ ∘ tail]
                           ∧ blockedThrds′  = blockedThrds \ {[here ↦ here, tid ↦ tid]}
                           ∧ runningThrds′ = runningThrds ∪ {[here ↦ here, tid ↦ tid]}
                           ∧ waitForMsgs′ = waitForMsgs \ {masterWFM}
                           ∧ C!IncrMSEQ(1)
                      ELSE ∧ C!ReplaceMsg(msg, [ mid        ↦ seq.mseq,
                                                 src        ↦ here,
                                                 dst        ↦ C!GetBackup(rootPlace),
                                                 source     ↦ here,
                                                 target     ↦ asyncDst,
                                                 fid        ↦ root,
                                                 finishSrc  ↦ finishSrc,
                                                 type       ↦ "backupGetNewMaster",
                                                 taskFID    ↦ msg.taskFID,
                                                 actionType ↦ "transit",
                                                 finishEnd  ↦ FALSE])
                           ∧ thrds′ = thrds
                           ∧ blockedThrds′  = blockedThrds
                           ∧ runningThrds′ = runningThrds
                           ∧ waitForMsgs′ = waitForMsgs \ {masterWFM}
                              we don't expect the backup to die
                              that is why we don't add
                              backupGetAdopterDone in waitForMsgs
                           ∧ C!IncrMSEQ(1)
                 ∧ IF backupPlace = BACKUP[fstates[root].here]
                      THEN fstates′ = fstates
                      ELSE fstates′ = [fstates EXCEPT ![fid].newBackup = backupPlace]
  ∧ UNCHANGED ⟨convFromDead, convToDead, mastersStatus, pstate, killed,
                 fmasters, fbackups⟩

MasterCompletedDone ≜
  ∧ pstate = "running"
  ∧ msgs ≠ {}
  ∧ LET msg ≜ C!FindMSG("masterCompletedDone")
     IN   ∧ msg ≠ C!NotMessage
          ∧ LET here ≜ msg.dst
                taskFID ≜ msg.taskFID
                root ≜ msg.fid
                success ≜ msg.success
                rootPlace ≜ C!GetFinishHome(root)
                backupPlace ≜ msg.backupPlace
                finishEnd ≜ msg.finishEnd
                source ≜ msg.source
```

```
        masterWFM ≜ [ src      ↦ rootPlace,
                      dst      ↦ here,
                      source   ↦ source,
                      target   ↦ here,
                      fid      ↦ root,
                      taskFID  ↦ msg.taskFID,
                      type     ↦ "masterCompletedDone" ]
        backupWFM ≜ [ src      ↦ backupPlace,
                      dst      ↦ here,
                      source   ↦ source,
                      target   ↦ here,
                      fid      ↦ root,
                      type     ↦ "backupCompletedDone" ]
IN      ∧ SetActionNameAndDepth(⟨"MasterCompletedDone", here⟩)
          Technically, we should check the condition rootPlace ∉ killed
          if success is true. we should communicate with the backup normally.
          the backup then should reject the request and notify the requester
          that the master has changed, so that we redirect the call to the
          new master.
        ∧ IF success
           THEN ∧ C!ReplaceMsg(msg, [mid          ↦ seq.mseq,
                                     src          ↦ here,
                                     dst          ↦ backupPlace,
                                     source       ↦ source,
                                     target       ↦ here,
                                     fid          ↦ root,
                                     knownMaster  ↦ msg.src,
                                     taskFID      ↦ taskFID,
                                     type         ↦ "backupCompleted",
                                     finishEnd    ↦ finishEnd])
                ∧ IF finishEnd THEN waitForMsgs' = (waitForMsgs \ {masterWFM})
                               ELSE waitForMsgs' = (waitForMsgs \ {masterWFM})
                                                     ∪ {backupWFM}
                ∧ C!IncrMSEQ(1)
           ELSE ∧ C!ReplaceMsg(msg, [ mid         ↦ seq.mseq,
                                      src         ↦ here,
                                      dst         ↦ C!GetBackup(rootPlace),
                                      source      ↦ msg.source,
                                      target      ↦ here,
                                      fid         ↦ root,
                                      taskFID     ↦ msg.taskFID,
                                      type        ↦ "backupGetNewMaster",
                                      finishEnd   ↦ FALSE,
                                      finishSrc   ↦ C!NotPlace,
                                      actionType  ↦ "completed"])
```

```
                    $\land\ waitForMsgs' = waitForMsgs \setminus \{masterWFM\}$
                        we don't expect backup to die
                        so we don't add $backupGetAdopterDone$
                        in $waitForMsgs$
                    $\land\ C!IncrMSEQ(1)$
                $\land$ IF $backupPlace = BACKUP[fstates[root].here]$
                    THEN $fstates' = fstates$
                    ELSE $fstates' = [fstates$ EXCEPT $![taskFID].newBackup = backupPlace]$
    $\land$ UNCHANGED $\langle convFromDead, convToDead, mastersStatus, pstate,$
                $thrds, killed, fmasters, fbackups,$
                $blockedThrds, runningThrds\rangle$
```

```
$BackupGetNewMasterDone \triangleq$
  $\land\ pstate =$ "running"
  $\land\ msgs \neq \{\}$
  $\land$ LET $msg \triangleq C!FindMSG($"backupGetNewMasterDone"$)$
     IN  $\land\ msg \neq C!NotMessage$
         $\land$ LET $here \triangleq msg.dst$
                 $actionType \triangleq msg.actionType$
                 $newMaster \triangleq msg.newMaster$
            IN  $\land\ SetActionNameAndDepth(\langle$"BackupGetNewMasterDone"$, here\rangle)$
                $\land$ IF $actionType =$ "transit"
                    THEN $\land\ C!ReplaceMsg(msg, [mid \mapsto seq.mseq,$
                                               $src \mapsto here,$
                                               $dst \mapsto newMaster,$
                                               $target \mapsto msg.target,$
                                               $fid \mapsto msg.fid,$
                                               $finishSrc \mapsto msg.finishSrc,$
                                               $taskFID \mapsto msg.taskFID,$
                                               $type \mapsto$ "masterTransit"$])$
                         $\land\ C!IncrMSEQ(1)$
                         $\land\ fstates' = [fstates$ EXCEPT $![msg.fid].newMaster = newMaster]$
                    ELSE IF $actionType =$ "completed"
                    THEN $\land\ C!ReplaceMsg(msg, [mid \mapsto seq.mseq,$
                                               $src \mapsto here,$
                                               $dst \mapsto newMaster,$
                                               $source \mapsto msg.source,$
                                               $target \mapsto msg.target,$
                                               $fid \mapsto msg.fid,$
                                               $finishEnd \mapsto msg.finishEnd,$
                                               $taskFID \mapsto msg.taskFID,$
                                               $type \mapsto$ "masterCompleted"$])$
                         $\land\ C!IncrMSEQ(1)$
                         $\land\ fstates' = [fstates$ EXCEPT $![msg.fid].newMaster = newMaster]$
                ELSE FALSE
  ∧ UNCHANGED ⟨pstate, thrds, killed, fmasters, fbackups, waitForMsgs,
                convFromDead, convToDead, mastersStatus, blockedThrds, runningThrds⟩
```

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

Terminated finish unblocks its thread

```
UnblockTerminateAsync ≜
  ∧ pstate = "running"
  ∧ LET pthrd ≜ FindBlockedThreadAsyncTerm
    IN  ∧ pthrd ≠ C!NotPlaceThread
        ∧ LET here ≜ pthrd.here
              tid ≜ pthrd.tid
              msg ≜ C!FindIncomingMSG(here, "backupCompletedDone")
              success ≜ msg.success
              top ≜ Head(thrds[here][tid].stack)
              blk ≜ top.b
              fid ≜ top.fid
              root ≜ msg.fid
              rootPlace ≜ C!GetFinishHome(root)
          IN  ∧ SetActionNameAndDepth(⟨"UnblockTerminateAsync", here,
                                        "success", success⟩)
              ∧ waitForMsgs' = waitForMsgs \ {[src    ↦ rootPlace,
                                               dst    ↦ here,
                                               target ↦ here,
                                               fid    ↦ root,
                                               type   ↦ "backupCompletedDone" ]}
              ∧ IF   Len(thrds[here][tid].stack) = 1
                THEN ∧ thrds' = [thrds EXCEPT ![here][tid].stack = ⟨⟩,
                                              ![here][tid].status = "idle"]
                     ∧ blockedThrds' = blockedThrds \ {[here ↦ here, tid ↦ tid]}
                     ∧ runningThrds' = runningThrds
              ELSE  $\land\ thrds' = [thrds$ EXCEPT $![here][tid].stack = Tail(@),$
                                                  $![here][tid].status =$ "running"$]$
                    $\land\ blockedThrds' = blockedThrds \setminus \{[here \mapsto here,\ tid \mapsto tid]\}$
                    $\land\ runningThrds' = runningThrds \cup \{[here \mapsto here,\ tid \mapsto tid]\}$
           $\land$ IF $blk = 0$
              THEN $pstate' =$ "terminated"
              ELSE $pstate' = pstate$
           $\land\ C!RecvMsg(msg)$
  $\land$ UNCHANGED $\langle convFromDead,\ convToDead,\ mastersStatus,\ fstates,\ seq,$
           $killed,\ fmasters,\ fbackups\rangle$
```

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

$AuthorizeTransitAsync \triangleq$
  $\land\ pstate =$ "running"
  $\land\ msgs \neq \{\}$
  $\land$ LET $pthrd \triangleq FindBlockedThreadAuthorizeTransitAsync$
     IN  $\land\ pthrd \neq C!NotPlaceThread$
         $\land$ LET $here \triangleq pthrd.here$
                 $tid \triangleq pthrd.tid$
                 $msg \triangleq C!FindIncomingMSG(here,$ "backupTransitDone"$)$
                 $success \triangleq msg.success$
                 $top \triangleq Head(thrds[here][tid].stack)$
                 $tail \triangleq Tail(thrds[here][tid].stack)$
                 $lstStmt \triangleq top.i$
                 $curStmt \triangleq top.i + 1$
                 $blk \triangleq top.b$
                 $root \triangleq msg.fid$
                 $fid \triangleq top.fid$
                 $rootPlace \triangleq C!GetFinishHome(root)$
                 $backupPlace \triangleq msg.src$
                 $nested \triangleq PROG[blk].stmts[curStmt]$
                 $asyncDst \triangleq PROG[nested].dst$
                 $realFID \triangleq root$
            IN  $\land\ SetActionNameAndDepth(\langle$"AuthorizeTransitAsync"$,\ here,$ "to"$,$
                                             $asyncDst,$ "success"$,\ success\rangle)$
                $\land\ C!ReplaceMsg(msg,$
                    [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,
                                              src ↦ top.src]
                                           ⟩ ∘ 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,
                                           finishSrc ↦ msg.finishSrc,
                                           target    ↦ asyncDst]}
  ∧ UNCHANGED ⟨convFromDead, convToDead, mastersStatus, fstates, pstate,
               killed, fmasters, fbackups⟩
```

```
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    ∧ 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 ⟨convFromDead, convToDead, mastersStatus, seq,
                killed, 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
          $finishSrc \triangleq msg.finishSrc$
          $finishHome \triangleq$ IF $fmasters[fid].home = C!NotPlace$
                                 THEN $here$
                                 ELSE $fmasters[fid].home$
          $backupPlace \triangleq$ IF $fmasters[fid].newBackup = C!NotPlace$
                                  THEN $BACKUP[finishHome]$
                                  ELSE $fmasters[fid].newBackup$
      IN  $\land SetActionNameAndDepth(\langle$"MasterTransit", $here\rangle)$
          $\land mastersStatus[here].status =$ "running"
          $\land src \neq C!NotPlace$
          $\land fid \neq C!NotID$
          $\land$ 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 $![fid].id = fid,$
                                                                $![fid].transit[src][target] = @ + 1,$
                                                                $![fid].transit[here][here] = 1,$
                                                                $![fid].numActive = @ + 2,$
                                                                $![fid].src = finishSrc,$
                                                                $![fid].home = src]$
                             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 $\land C!ReplaceMsg(msg,$ [ $mid \mapsto seq.mseq,$
                                                $src \mapsto here,$
                                                $dst \mapsto src,$
                                             $target \mapsto target,$
                                                $fid \mapsto fid,$
                                            $taskFID \mapsto msg.taskFID,$
                                               $type \mapsto$ "masterTransitDone",
                                             $submit \mapsto submit,$
                                            $success \mapsto$ TRUE,
                                          $finishSrc \mapsto finishSrc,$
                                        $backupPlace \mapsto backupPlace])$
                          $\land C!IncrMSEQ(1)$
  $\land$ UNCHANGED $\langle waitForMsgs, convFromDead, convToDead, mastersStatus,$
                $fstates, pstate, thrds, killed, fbackups,$
                $blockedThrds, runningThrds\rangle$
```

```
$MasterCompleted \triangleq$
  $\land pstate =$ "running"
```

```
$\land\ msgs \neq \{\}$
$\land$ LET $msg \triangleq C!FindMSG($“masterCompleted”$)$
    IN  $\land\ msg \neq C!NotMessage$
        $\land$ LET $here \triangleq msg.dst$
                $mid \triangleq msg.mid$
                $fid \triangleq msg.fid$
                $src \triangleq msg.src$
                $source \triangleq msg.source$
                $target \triangleq msg.target$
                $finishEnd \triangleq msg.finishEnd$
                $finishHome \triangleq$ IF $fmasters[fid].home = C!NotPlace$
                                 THEN $here$
                                 ELSE $fmasters[fid].home$
                $backupPlace \triangleq$ IF $fmasters[fid].newBackup = C!NotPlace$
                                  THEN $BACKUP[finishHome]$
                                  ELSE $fmasters[fid].newBackup$
                $releaseMSG \triangleq [mid \mapsto seq.mseq,$
                                 $src \mapsto here,$
                                 $dst \mapsto here,$
                                 $fid \mapsto fid,$
                                 $type \mapsto$ “releaseFinish”$]$
                $completedDone \triangleq [mid \mapsto seq.mseq + 1,$
                                    $src \mapsto here,$
                                    $dst \mapsto src,$
                                    $source \mapsto source,$
                                    $target \mapsto target,$
                                    $fid \mapsto fid,$
                                    $taskFID \mapsto msg.taskFID,$
                                    $type \mapsto$ “masterCompletedDone”,
                                    $success \mapsto$ TRUE,
                                    $finishEnd \mapsto finishEnd,$
                                    $backupPlace \mapsto backupPlace]$
                $adopterCompleted \triangleq [mid \mapsto seq.mseq + 2,$
                                       $src \mapsto here,$
                                       $dst \mapsto fmasters[fid].adopterPlace,$
                                       $source \mapsto fmasters[fid].src,$
                                       $target \mapsto fmasters[fid].home,$
                                       $fid \mapsto fstates[fid].eroot,$
                                       $taskFID \mapsto fstates[fid].eroot,$
                                       $finishEnd \mapsto$ FALSE,
                                       $type \mapsto$ “masterCompleted”$]$
            IN  $\land\ SetActionNameAndDepth(\langle$“MasterCompleted”, $here$, “fid”, $fid$, “home”, $finishHome\rangle)$
                $\land\ backupPlace \neq C!NotPlace$
                $\land\ fid \neq C!NotID$
                 $\land\ target = src$ we cannot check this because the $src$ can be the $newMaster$ of a lost finish
            ∧ $mastersStatus[here].status =$ "running"
            ∧ IF $(fmasters[fid].transit[source][target] > 0)$
               THEN ∧ $fmasters' = [fmasters$ EXCEPT $![fid].transit[source][target] = @ - 1,$
                                                    $![fid].numActive \quad = @ - 1,$
                                                    $![fid].isReleased \quad =$
                                                        $(fmasters[fid].numActive = 1)]$
               ELSE ∧ $target \in killed$
                    ∧ $fmasters' = fmasters$
            ∧ IF $(fmasters'[fid].numActive = 0 \land src \notin killed)$
               THEN ∧ IF $fmasters[fid].isAdopted$
                         THEN ∧ $C!ReplaceMsgSet(msg, \{completedDone, adopterCompleted\})$
                         ELSE ∧ $C!ReplaceMsgSet(msg, \{releaseMSG, completedDone\})$
                    ∧ $C!IncrMSEQ(3)$
               ELSE IF $fmasters'[fid].numActive = 0$
               THEN ∧ IF $fmasters[fid].isAdopted$
                         THEN ∧ $C!ReplaceMsg(msg, adopterCompleted)$
                         ELSE ∧ $C!ReplaceMsg(msg, releaseMSG)$
                    ∧ $C!IncrMSEQ(3)$
               ELSE IF $src \notin killed$
               THEN ∧ $C!ReplaceMsg(msg, completedDone)$
                    ∧ $C!IncrMSEQ(3)$
               ELSE ∧ $C!RecvMsg(msg)$
                    ∧ $seq' = seq$
  ∧ UNCHANGED $\langle convFromDead, convToDead, mastersStatus, fstates, pstate,$
              $thrds, killed, fbackups, waitForMsgs,$
              $blockedThrds, runningThrds\rangle$
```

Finish backup replica actions

```
BackupGetNewMaster ≜
  ∧ $pstate =$ "running"
  ∧ $msgs \neq \{\}$
  ∧ LET $msg \triangleq C!FindMSG($"backupGetNewMaster"$)$
     IN  ∧ $msg \neq C!NotMessage$
         ∧ 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  ∧ $SetActionNameAndDepth(\langle$"backupGetNewMaster"$, here\rangle)$
               ∧ $fbackups[fid].newMaster \neq C!NotPlace$
               ∧ IF $src \in killed \lor msg.dst \in killed$
                  THEN ∧ $C!RecvMsg(msg)$
                       ∧ $seq' = seq$
```

```
                        ELSE  ∧ $C!ReplaceMsg(msg, [$  $mid \mapsto seq.mseq,$
                                                       $src \mapsto here,$
                                                       $dst \mapsto src,$
                                                       $source \mapsto source,$
                                                       $target \mapsto target,$
                                                       $fid \mapsto fid,$
                                                       $newMaster \mapsto fbackups[fid].newMaster,$
                                                       $actionType \mapsto actionType,$
                                                       $finishEnd \mapsto msg.finishEnd,$
                                                       $finishSrc \mapsto msg.finishSrc,$
                                                       $taskFID \mapsto msg.taskFID,$
                                                       $type \mapsto$ "backupGetNewMasterDone"$])$
                              ∧ $C!IncrMSEQ(1)$
  ∧ UNCHANGED $\langle fstates, pstate, thrds, killed, fmasters,$
              $fbackups, waitForMsgs, mastersStatus, convFromDead, convToDead,$
              $blockedThrds, runningThrds\rangle$

$BackupTransit \triangleq$
  ∧ $pstate =$ "running"
  ∧ $msgs \neq \{\}$
  ∧ LET $msg \triangleq C!FindMSG($"backupTransit"$)$
    IN  ∧ $msg \neq C!NotMessage$
        ∧ LET $here \triangleq msg.dst$
              $fid \triangleq msg.fid$
              $src \triangleq msg.src$
              $target \triangleq msg.target$
              $finishSrc \triangleq msg.finishSrc$
              $knownMaster \triangleq msg.knownMaster$
              $correctBackup \triangleq$ IF $fmasters[fid].newBackup \neq C!NotPlace$
                                       THEN $fmasters[fid].newBackup$
                                       ELSE $BACKUP[fstates[fid].here]$
              $backupKnownMaster \triangleq$ IF $fbackups[fid].newMaster \neq C!NotPlace$
                                           THEN $fbackups[fid].newMaster$
                                           ELSE $fstates[fid].here$
              $masterChanged \triangleq backupKnownMaster \neq knownMaster$
          IN  ∧ $SetActionNameAndDepth(\langle$"BackupTransit"$, here\rangle)$
              ∧ $correctBackup = here$
              ∧ IF $masterChanged$
                  THEN $fbackups' = fbackups$
                  ELSE IF $fbackups[fid].id = C!NotID$
                  THEN $fbackups' = [fbackups$ EXCEPT $![fid].id = fid,$
                                                      $![fid].transit[src][target] = @ + 1,$
                                                      $![fid].transit[src][src] = 1,$
                                                      $![fid].numActive = @ + 2,$
                                                      $![fid].src = finishSrc,$
```

```
                                                  $![fid].home = src]$
                ELSE  $fbackups' = [fbackups$ EXCEPT $![fid].transit[src][target] = @ + 1,$
                                                  $![fid].numActive\ =\ @ + 1]$
             $\wedge$ IF $src \in killed$
                THEN  $\wedge\ C!RecvMsg(msg)$
                      $\wedge\ seq' = seq$
                ELSE  IF $masterChanged$
                THEN  $\wedge\ C!ReplaceMsg(msg, [\ mid \mapsto seq.mseq,$
                                           $src \mapsto here,$
                                           $dst \mapsto src,$
                                           $target \mapsto target,$
                                           $fid \mapsto fid,$
                                           $taskFID \mapsto msg.taskFID,$
                                           $type \mapsto$ "masterTransitDone",
                                           $submit \mapsto$ FALSE,
                                           $success \mapsto$ FALSE,
                                           $finishSrc \mapsto finishSrc,$
                                           $backupPlace \mapsto here])$
                      $\wedge\ C!IncrMSEQ(1)$
                ELSE  $\wedge\ C!ReplaceMsg(msg, [\ mid \mapsto seq.mseq,$
                                           $src \mapsto here,$
                                           $dst \mapsto src,$
                                           $target \mapsto target,$
                                           $fid \mapsto fid,$
                                           $finishSrc \mapsto finishSrc,$
                                           $type \mapsto$ "backupTransitDone",
                                           $success \mapsto$ TRUE])
                      $\wedge\ C!IncrMSEQ(1)$
  $\wedge$ UNCHANGED $\langle convFromDead, convToDead, mastersStatus, fstates, pstate,$
                 $thrds, killed, fmasters, waitForMsgs,$
                 $blockedThrds, runningThrds\rangle$

$BackupCompleted \triangleq$
  $\wedge\ pstate =$ "running"
  $\wedge\ msgs \neq \{\}$
  $\wedge$ LET $msg \triangleq C!FindMSG($"backupCompleted"$)$
     IN   $\wedge\ msg \neq C!NotMessage$
          $\wedge$ LET $here \triangleq msg.dst$
                 $fid \triangleq msg.fid$
                 $src \triangleq msg.src$
                 $source \triangleq msg.source$
                 $target \triangleq msg.target$
                 $finishEnd \triangleq msg.finishEnd$
                 $knownMaster \triangleq msg.knownMaster$
                 $correctBackup \triangleq$ IF $fmasters[fid].newBackup \neq C!NotPlace$
```

```
                              THEN $fmasters[fid].newBackup$
                              ELSE $BACKUP[fstates[fid].here]$
        $backupKnownMaster \triangleq$ IF $fbackups[fid].newMaster \neq C!NotPlace$
                                 THEN $fbackups[fid].newMaster$
                                 ELSE $fstates[fid].here$
        $masterChanged \triangleq backupKnownMaster \neq knownMaster$
    IN  $\land SetActionNameAndDepth(\langle$"BackupCompleted"$, here\rangle)$
        $\land correctBackup = here$
        $\land fbackups[fid].transit[source][target] > 0$
        $\land fbackups[fid].numActive > 0$
        $\land$ IF $masterChanged$
             THEN $fbackups' = fbackups$
             ELSE $fbackups' = [fbackups$ EXCEPT $![fid].transit[source][target] = @ - 1,$
                                                $!\ [fid].numActive = @ - 1]$
        $\land$ IF $src \in killed \lor finishEnd$
             THEN $\land C!RecvMsg(msg)$
                  $\land seq' = seq$
             ELSE IF $masterChanged$
             THEN $\land C!ReplaceMsg(msg,$ $[mid \mapsto seq.mseq,$
                                             $src \mapsto here,$
                                             $dst \mapsto src,$
                                             $source \mapsto source,$
                                             $target \mapsto target,$
                                             $fid \mapsto fid,$
                                             $taskFID \mapsto msg.taskFID,$
                                             $type \mapsto$ "masterCompletedDone",
                                             $success \mapsto$ FALSE,
                                             $finishEnd \mapsto finishEnd,$
                                             $backupPlace \mapsto here])$
                  $\land C!IncrMSEQ(1)$
             ELSE $\land C!ReplaceMsg(msg,$ $[mid \mapsto seq.mseq,$
                                             $src \mapsto here,$
                                             $dst \mapsto src,$
                                             $target \mapsto target,$
                                             $fid \mapsto fid,$
                                             $type \mapsto$ "backupCompletedDone",
                                             $success \mapsto$ TRUE$])$
                  $\land C!IncrMSEQ(1)$
$\land$ UNCHANGED $\langle convFromDead, convToDead, mastersStatus, fstates, pstate,$
                  $thrds, killed, fmasters, waitForMsgs,$
                  $blockedThrds, runningThrds\rangle$
```

Recovery actions

```
$PrepareConvertTasks \triangleq$
  $\land\ pstate = \text{“running”}$
  $\land\ \exists\, p \in PLACE : mastersStatus[p].status = \text{“preConvert”}$
  $\land$ LET $pset \triangleq \{p \in PLACE :$
                     $\land\ mastersStatus[p].status = \text{“preConvert”}$
                     $\land\ p \notin killed\}$
          $here \triangleq$ IF $pset = \{\}$ THEN $C!NotPlace$ ELSE CHOOSE $p \in pset :$ TRUE
          $dead \triangleq mastersStatus[here].lastKilled$
     IN   $\land\ SetActionNameAndDepth(\langle \text{“PrepareConvertTasks”}, here\rangle)$
          $\land\ here \neq C!NotPlace$
          $\land\ convFromDead' = convFromDead \cup \{t \in C!ConvTask :$
                                      $\land\ t.to\_pl \neq C!NotPlace$
                                      $\land\ t.to\_pl \neq dead$
                                      $\land\ t.to\_pl \notin killed$
                                      $\land\ t.from\_pl = dead$
                                      $\land\ t.fid \in \{id \in C!IDRange :$
                                            $\land\ fmasters[id].id \neq C!NotID$
                                            $\land\ \lor\ fstates[id].here = here$
                                              $\lor\ \land\ fstates[id].here = dead$
                                                $\land\ fbackups[id].newMaster = here$
                                            $\land\ fmasters[id].transit[t.from\_pl][t.to\_pl] > 0\}$
                                      $\land\ t.here =$ IF $fstates[t.fid].here \neq dead$
                                               THEN $fstates[t.fid].here$
                                               ELSE $fbackups[t.fid].newMaster\}$
          $\land\ convToDead' = convToDead \cup \{t \in C!ConvTask :$
                                  $\land\ t.from\_pl \neq C!NotPlace$
                                  $\land\ t.to\_pl = dead$
                                  $\land\ t.fid \in \{id \in C!IDRange :$
                                        $\land\ fmasters[id].id \neq C!NotID$
                                        $\land\ \lor\ fstates[id].here = here$
                                          $\lor\ \land\ fstates[id].here = dead$
                                            $\land\ fbackups[id].newMaster = here$
                                        $\land\ fmasters[id].transit[t.from\_pl][t.to\_pl] > 0\}$
                                  $\land\ t.here =$ IF $fstates[t.fid].here \neq dead$
                                           THEN $fstates[t.fid].here$
                                           ELSE $fbackups[t.fid].newMaster\}$
          $\land\ mastersStatus' = [mastersStatus$ EXCEPT $![here].status =$
                                    IF $\exists\, m \in convToDead' : m.here = here$
                                     THEN “convertToDead”
                                     ELSE IF $\exists\, m \in convFromDead' : m.here = here$
                                     THEN “convertFromDead”
                                     ELSE “running”]
  $\land$ UNCHANGED $\langle fstates, msgs, pstate, seq, thrds, killed, fmasters, fbackups, waitForMsgs,$
                    $blockedThrds, runningThrds\rangle$
```

```
$GetConvertToDeadSeeker \triangleq$
  IF $convToDead = \{\}$ THEN $C!NotConvTask$
   ELSE CHOOSE $m \in convToDead : mastersStatus[m.here].status =$ “convertToDead”

$ConvertToDead \triangleq$
  $\land\ pstate =$ “running”
  $\land\ \exists\, p \in PLACE : mastersStatus[p].status =$ “convertToDead”
  $\land$ LET $convSeeker \triangleq GetConvertToDeadSeeker$
     IN  $\land\ convSeeker \neq C!NotConvTask$
         $\land\ convSeeker.here \notin killed$
         $\land$ LET $here \triangleq convSeeker.here$
                 $source \triangleq convSeeker.from\_pl$
                 $fid \triangleq convSeeker.fid$
                 $target \triangleq convSeeker.to\_pl$ dead place
                 $backups \triangleq [r \in C!IDRange \mapsto$ IF $\land\ fbackups[r].src = source$
                                                          $\land\ fstates[r].eroot = fid$
                                                          $\land\ fbackups[r].home = target$
                                                      THEN $1$
                                                      ELSE $0]$
                 $adoptedChildren \triangleq \{f \in C!IDRange : backups[f] = 1\}$
                 $t1 \triangleq fmasters[fid].transit[source][target]$
                  a workaround to get the set size assuming it doesn't exceed 5
                 $t2 \triangleq$ CHOOSE $x \in 0\,..\,5 : x = backups[1] + backups[2] + backups[3] +$
                                                  $backups[4] + backups[5]$
                 $releaseMSG \triangleq [mid \mapsto seq.mseq,$
                                 $src \mapsto here,$
                                 $dst \mapsto here,$
                                 $fid \mapsto fid,$
                                 $type \mapsto$ “releaseFinish” $]$
                 $adopterCompleted \triangleq [mid \mapsto seq.mseq + 1,$
                                       $src \mapsto here,$
                                       $dst \mapsto fmasters'[fid].adopterPlace,$
                                    $source \mapsto fmasters[fid].src,$
                                    $target \mapsto fmasters[fid].home,$
                                       $fid \mapsto fstates[fid].eroot,$
                                   $taskFID \mapsto fstates[fid].eroot,$
                                 $finishEnd \mapsto$ FALSE,
                                      $type \mapsto$ “masterCompleted” $]$
           IN  $\land\ SetActionNameAndDepth(\langle$“ConvertToDead”, $here$, “t1”, $t1$, “t2”, $t2\rangle)$
               $\land\ convToDead' = convToDead \setminus \{convSeeker\}$
               $\land\ target = mastersStatus[here].lastKilled$
               $\land\ t1 \geq t2$
               $\land\ t1 > 0$
               $\land\ fmasters' = [r \in C!IDRange \mapsto$
```

```
                          IF r = fid
                           THEN [fmasters[r] EXCEPT !.numActive = @ − (t1 − t2),
                                                    !.transit = [@ EXCEPT ![source][target] = t2],
                                                    !.adoptedChildren = adoptedChildren]
                           ELSE IF r ∈ adoptedChildren
                           THEN [fmasters[r] EXCEPT !.isAdopted = TRUE,
                                                    !.adopterPlace = here]
                           ELSE fmasters[r]]
                ∧ fbackups′ = [r ∈ C!IDRange ↦
                          IF r ∈ adoptedChildren
                           THEN [fbackups[r] EXCEPT !.isAdopted = TRUE,
                                                    !.adopterPlace = here]
                           ELSE fbackups[r] ]
                ∧ IF fmasters′[fid].numActive = 0
                    THEN IF fmasters′[fid].isAdopted
                           THEN ∧ C!SendMsg(adopterCompleted)
                                ∧ C!IncrMSEQ(1)
                           ELSE ∧ C!SendMsg(releaseMSG)
                                ∧ C!IncrMSEQ(1)
                    ELSE ∧ msgs′ = msgs
                         ∧ seq′ = seq
                ∧ IF ∃ m ∈ convToDead′ : m.here = here
                    THEN mastersStatus′ = mastersStatus
                    ELSE IF ∃ m ∈ convFromDead : m.here = here
                    THEN mastersStatus′ = [mastersStatus EXCEPT ![here].status = "convertFromDead"]
                    ELSE mastersStatus′ = [mastersStatus EXCEPT ![here].status = "running"]
  ∧ UNCHANGED ⟨fstates, pstate, thrds, killed, waitForMsgs, convFromDead,
               blockedThrds, runningThrds⟩
```

```
GetConvertFromDeadSeeker ≜
  IF convFromDead = {} THEN C!NotConvTask
   ELSE CHOOSE m ∈ convFromDead : mastersStatus[m.here].status = "convertFromDead"

ConvertFromDead ≜
  ∧ pstate = "running"
  ∧ ∃ p ∈ PLACE : mastersStatus[p].status = "convertFromDead"
  ∧ LET convSeeker ≜ GetConvertFromDeadSeeker
     IN  ∧ convSeeker ≠ C!NotConvTask
         ∧ convSeeker.here ∉ killed
         ∧ LET here ≜ convSeeker.here
               source ≜ convSeeker.from_pl   dead place
               fid ≜ convSeeker.fid
               target ≜ convSeeker.to_pl
               remotes ≜ {f ∈ C!IDRange :
                    $\land fstates[f].type = $ "distremote"
                    $\land fstates[f].root = fid$
                    $\land fstates[f].here = target\}$
          $remFID \triangleq$ IF $remotes = \{\}$
                              THEN $C!NotID$
                              ELSE CHOOSE $r \in remotes :$ TRUE
          $t1 \triangleq fmasters[fid].transit[source][target]$
          $t2 \triangleq$ IF $remFID = C!NotID$
                    THEN $0$
                    ELSE $fstates[remFID].received[source]$
          $releaseMSG \triangleq [mid \mapsto seq.mseq,$
                             $src \mapsto here,$
                             $dst \mapsto here,$
                             $fid \mapsto fid,$
                             $type \mapsto$ "releaseFinish"$]$
          $adopterCompleted \triangleq [mid \mapsto seq.mseq + 1,$
                                   $src \mapsto here,$
                                   $dst \mapsto fmasters[fid].adopterPlace,$
                                $source \mapsto fmasters[fid].src,$
                                $target \mapsto fmasters[fid].home,$
                                   $fid \mapsto fstates[fid].eroot,$
                               $taskFID \mapsto fstates[fid].eroot,$
                             $finishEnd \mapsto$ FALSE,
                                 $type \mapsto$ "masterCompleted"$]$
      IN  $\land SetActionNameAndDepth(\langle$"ConvertFromDead", $here,$ "remFID", $remFID,$ "t1", $t1,$ "t2", $t2$
          $\land convFromDead' = convFromDead \setminus \{convSeeker\}$
          $\land t1 \geq t2$
          $\land t1 > 0$
          $\land fmasters' = [fmasters$ EXCEPT $![fid].numActive = @ - (t1 - t2),$
                                              $![fid].transit[source][target] = t2]$
          $\land$ IF $remFID \neq C!NotID$
              THEN $fstates' = [fstates$ EXCEPT $![remFID].deny = @ \cup \{source\}]$
              ELSE $fstates' = fstates$
          $\land$ IF $fmasters'[fid].numActive = 0$
              THEN IF $fmasters[fid].isAdopted$
                     THEN $\land C!SendMsg(adopterCompleted)$
                          $\land C!IncrMSEQ(1)$
                     ELSE $\land C!SendMsg(releaseMSG)$
                          $\land C!IncrMSEQ(1)$
              ELSE $\land msgs' = msgs$
                   $\land seq' = seq$
          $\land$ IF $\exists m \in convFromDead' : m.here = here$
              THEN $mastersStatus' = mastersStatus$
              ELSE $mastersStatus' = [mastersStatus$ EXCEPT $![here].status =$ "running"$]$
$\land$ UNCHANGED $\langle pstate, thrds, killed, fbackups, waitForMsgs, convToDead,$
```

```
                    $blockedThrds,\ runningThrds\rangle$

$FindWaitForMSG \triangleq$
  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 $msg.type =$ "masterCompletedDone"
                      THEN IF $\neg(\exists\, m \in msgs :$  message has been sent already
                                 $\land\ m.type = msg.type \land m.src = msg.src$
                                 $\land\ m.dst = msg.dst \land m.fid = msg.fid$
                                 $\land\ m.source = msg.source$
                                 $\land\ m.target = msg.target$
                                 $\land\ m.taskFID = msg.taskFID$
                                 $\land\ m.success)$
                            THEN $\land\ msgs' = (msgs \setminus delMsgs) \cup \{$
                                          [ $mid \mapsto seq.mseq,$
                                            $src \mapsto msg.src,$
                                            $dst \mapsto msg.dst,$
                                            $source \mapsto msg.source,$
                                            $target \mapsto msg.target,$
                                            $fid \mapsto msg.fid,$
                                            $taskFID \mapsto msg.taskFID,$
                                            $type \mapsto$ "masterCompletedDone",
                                            $success \mapsto$ FALSE,
                                            $finishEnd \mapsto$ FALSE,
                                            $backupPlace \mapsto C!NotPlace]\}$
                            ELSE $\land\ msgs' = (msgs \setminus delMsgs)$
ELSE IF $msg.type = $ "masterTransitDone"
THEN IF $\neg(\exists\, m \in msgs :$ message has been sent already
            $\land\ m.type = msg.type \land m.src = msg.src$
            $\land\ m.dst = msg.dst \land m.fid = msg.fid$
            $\land\ m.taskFID = msg.taskFID$
            $\land\ m.success)$
     THEN  $\land\ msgs' = (msgs \setminus delMsgs) \cup \{$
                 $[\ mid \mapsto seq.mseq,$
                    $src \mapsto msg.src,$
                    $dst \mapsto msg.dst,$
                    $target \mapsto msg.target,$
                    $fid \mapsto msg.fid,$
                    $taskFID \mapsto msg.taskFID,$
                    $finishSrc \mapsto msg.finishSrc,$
                    $type \mapsto$ "masterTransitDone",
                    $backupPlace \mapsto C!NotPlace,$
                    $submit \mapsto$ FALSE,
                    $success \mapsto$ FALSE$]\}$
     ELSE  $\land\ msgs' = (msgs \setminus delMsgs)$
ELSE IF $msg.type = $ "backupCompletedDone"
THEN IF $\neg(\exists\, m \in msgs :$ message has been sent already
            $\land\ m.type = msg.type \land m.src = msg.src$
            $\land\ m.dst = msg.dst \land m.fid = msg.fid$
            $\land\ m.isAdopter = msg.isAdopter \land m.success)$
     THEN  $\land\ msgs' = (msgs \setminus delMsgs) \cup \{$
                 $[\ mid \mapsto seq.mseq,$
                    $src \mapsto msg.src,$
                    $dst \mapsto msg.dst,$
                    $target \mapsto msg.target,$
                    $fid \mapsto msg.fid,$
                    $type \mapsto$ "backupCompletedDone",
                    $success \mapsto$ FALSE$]\}$
     ELSE  $\land\ msgs' = (msgs \setminus delMsgs)$
ELSE IF $msg.type = $ "backupTransitDone"
THEN IF $\neg(\exists\, m \in msgs :$ message has been sent already
            $\land\ m.type = msg.type \land m.src = msg.src$
            $\land\ m.dst = msg.dst \land m.fid = msg.fid$
            $\land\ m.target = msg.target \land m.success)$
     THEN  $\land\ msgs' = (msgs \setminus delMsgs) \cup \{$
                 $[\ mid \mapsto seq.mseq,$
                    $src \mapsto msg.src,$
                    $dst \mapsto msg.dst,$
                    $target \mapsto msg.target,$
                    $finishSrc \mapsto msg.finishSrc,$
                    $fid \mapsto msg.fid,$
```

```
                                              type ↦ "backupTransitDone",
                                            success ↦ FALSE]}
                              ELSE  ∧ msgs' = (msgs \ delMsgs)
                         ELSE  FALSE
  ∧ UNCHANGED ⟨convFromDead, convToDead, mastersStatus, fstates, pstate,
               thrds, killed, fmasters, fbackups,
               blockedThrds, runningThrds⟩
```

Predicate enumerating all possible next actions

```
Next ≜
  ∨ RecvAsync
  ∨ ReleaseRootFinish
  ∨ BackupTransit
  ∨ BackupCompleted
  ∨ BackupGetNewMaster
  ∨ BackupGetNewMasterDone
  ∨ MasterTransit
  ∨ MasterCompleted
  ∨ MasterTransitDone
  ∨ MasterCompletedDone
  ∨ PrepareConvertTasks
  ∨ ConvertFromDead
  ∨ ConvertToDead
  ∨ SimulateFailedResponse
  ∨ RunExprOrKill
  ∨ ScheduleNestedFinish
  ∨ TerminateAsync
  ∨ SpawnRemoteAsync
  ∨ SpawnLocalAsync
  ∨ StopFinish
  ∨ StartFinish
  ∨ AuthorizeTransitAsync
  ∨ UnblockTerminateAsync
```

Asserting fairness properties to all actions

```
Liveness ≜
  ∧ WF_Vars(RecvAsync)
  ∧ WF_Vars(ReleaseRootFinish)
  ∧ WF_Vars(StartFinish)
  ∧ WF_Vars(StopFinish)
  ∧ WF_Vars(SpawnLocalAsync)
  ∧ WF_Vars(SpawnRemoteAsync)
  ∧ WF_Vars(TerminateAsync)
  ∧ WF_Vars(ScheduleNestedFinish)
```

$\land \mathrm{WF}_{Vars}(\mathit{RunExprOrKill})$
$\land \mathrm{WF}_{Vars}(\mathit{BackupTransit})$
$\land \mathrm{WF}_{Vars}(\mathit{BackupCompleted})$
$\land \mathrm{WF}_{Vars}(\mathit{MasterTransit})$
$\land \mathrm{WF}_{Vars}(\mathit{MasterCompleted})$
$\land \mathrm{WF}_{Vars}(\mathit{MasterTransitDone})$
$\land \mathrm{WF}_{Vars}(\mathit{MasterCompletedDone})$
$\land \mathrm{WF}_{Vars}(\mathit{PrepareConvertTasks})$
$\land \mathrm{WF}_{Vars}(\mathit{ConvertToDead})$
$\land \mathrm{WF}_{Vars}(\mathit{ConvertFromDead})$
$\land \mathrm{WF}_{Vars}(\mathit{SimulateFailedResponse})$
$\land \mathrm{WF}_{Vars}(\mathit{BackupGetNewMaster})$
$\land \mathrm{WF}_{Vars}(\mathit{BackupGetNewMasterDone})$
$\land \mathrm{WF}_{Vars}(\mathit{AuthorizeTransitAsync})$
$\land \mathrm{WF}_{Vars}(\mathit{UnblockTerminateAsync})$

Specification

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

THEOREM $\mathit{Spec} \Rightarrow \Box(\mathit{TypeOK} \land \mathit{StateOK})$

\* Modification History
\* Last modified *Mon Dec* 18 10:19:13 *AEDT* 2017 by *u*5482878
\* Last modified *Wed Dec* 13 23:25:41 *AEDT* 2017 by *shamouda*
\* Created *Wed Sep* 13 12:14:43 *AEST* 2017 by *u*5482878