MODULE *Executor*

This specification models a subset of $X10$ programs that use finish, *async* at (place), simple expression statements, or error statements that raise exceptions. The following is a sample program that this specification can validate

```
finish {
 expr;
 async at(p1){
  expr;
  async at(p2){
   expr;
   error;
   expr;
  }
 }
 async at(p3){
  expr;
 }
}
```

Our goal is to ensure that the finish construct correctly detects termination and never causes a deadlock

EXTENDS *Integers*, *Sequences*

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
$ROOT\_FINISH$,	The selected finish implementation for root
$REMOTE\_FINISH$,	The selected finish implementation for remote
$NTHREADS$,	Minimum number of threads, more threads can be added up to $MXTHREADS$ when running threads blocks
$MXTHREADS$,	Maximum number of threads per place
$NBLOCKS$,	Number of blocks in input program
$MXSTMTS$,	Maximum number of statements within a block
$MUST\_NOT\_RUN$,	Validation constant: blocks that must not run, for example were not executed because of an execption
$MXDEAD$	Maximum number of places to kill

Variables common to all finish implementations

VARIABLES

$fstates$,	Array of finish states
$msgs$,	The set of inflight messages. We delete a message once received
$pstate$,	Program state: $init \rightarrow running \rightarrow terminated$

program,	Finish body as a sequence of statements
aseq,	Sequence to generate activity ids
fseq,	Sequence to generate finish ids
mseq,	Sequence to generate *msg* ids
readyQ,	Queue of ready activities at each place
thrds,	Threads at all places
incPar,	Increase parallelism requests
decPar,	Decrease parallelism requests
ppProgram,	Preprocessing temporary variable: program
ppcurStmt,	Preprocessing temporary variable: current statement
killed,	The set places killed so far
killedCnt,	Current number of killed places
pendingAct,	Set of activities received at destination place but need permission from the resilient store to run
isDead	the dead places recognized at each place

Variables used by *Place*0 resilient store (*P0ResStore* module)

VARIABLES

$p0state$,	The state of *place*0 resilient store
$p0fstates$,	*Place*0 resilient finish states
$p0dead$,	Last reported dead place
$p0adoptSet$,	set used in adoption phase 1: seek adoption
$p0convSet$	set used in adoption phase 2: convert dead tasks

$Vars \triangleq \langle fstates,\ msgs,\ pstate,\ program,\ aseq,\ fseq,\ mseq,\ p0dead,$
$readyQ,\ thrds,\ ppProgram,\ ppcurStmt,\ incPar,\ decPar,\ p0state,\ p0convSet,$
$killed,\ killedCnt,\ p0fstates,\ pendingAct,\ isDead,\ p0adoptSet\rangle$

Predicate to hide the finish implementation

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

INSTANCE $P0ResStore$

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

Invariants (formulas true in every reachable state.)

$TypeOK \triangleq$
$\land\ \ fstates \in [IDRange \to FinishState]$
$\land\ \ readyQ \in [PLACE \to Seq(Activity)]$
$\land\ \ thrds \in\ [PLACE\ \ \to [ThreadID \to Thread]]$

$\land$ $msgs \subseteq Messages$
$\land$ $pstate \in \{\text{“init”}, \text{“running”}, \text{“terminated”}, \text{“exceptionThrown”}\}$
$\land$ $program \in [BlockID \to Block]$
$\land$ $PROG\_HOME \in PLACE$
$\land$ $aseq \in Nat$
$\land$ $mseq \in Nat$
$\land$ $fseq \in IDRange$
$\land$ $ppcurStmt \in Nat$
$\land$ $incPar \in [PLACE \to Nat]$
$\land$ $decPar \in [PLACE \to Nat]$
$\land$ $MUST\_NOT\_RUN \subseteq BlockID$
$\land$ $MXDEAD \in Nat$
$\land$ IF $ROOT\_FINISH \in \{\text{“SPMDroot”}, \text{“SPMDremote”}, \text{“root”}, \text{“remote”}\}$
THEN $MXDEAD = 0$ ELSE TRUE don't kill places in non-resilient mode
$\land$ $killed \subseteq PLACE$
$\land$ $killedCnt \in Nat$
$\land$ $p0fstates \in [IDRange \to ResilientFinishState]$
$\land$ $pendingAct \subseteq Activity$
$\land$ $isDead \in [PLACE \to [PLACE \to \text{BOOLEAN }]]$
$\land$ $p0dead \in PLACE \cup \{NotPlace\}$
$\land$ $p0adoptSet \subseteq Adopter$
$\land$ $p0convSet \subseteq ConvTask$
$\land$ $p0state \in \{\text{“running”}, \text{“seekAdoption”}, \text{“convertDead”}, \text{“release”}\}$

$StateOK \triangleq$
$\lor$ $\land pstate \in \{\text{“init”}, \text{“running”}\}$
$\lor$ $\land pstate \in \{\text{“terminated”}, \text{“exceptionThrown”}\}$
$\land ppProgram = \langle\rangle$
$\land msgs = \{\}$
$\land \forall p \in PLACE :$
$\land readyQ[p] = \langle\rangle$
$\land \forall t \in ThreadID : thrds[p][t].stack = \langle\rangle$
$\land \forall fid \in IDRange :$
$\land fstates[fid].status \in \{\text{“unused”}, \text{“forgotten”}\}$
$\land$ IF $pstate =$ “terminated”
THEN $\land fstates[FIRST\_ID].excs = \langle\rangle$
$\land \forall b \in BlockID : program[b].ran = \text{TRUE} \lor program[b].b = NotBlockID$
ELSE $\land fstates[FIRST\_ID].excs \neq \langle\rangle$
$\land \forall b \in BlockID :$ IF $b \in MUST\_NOT\_RUN$
THEN $program[b].ran = \text{FALSE}$
ELSE $program[b].ran = \text{TRUE}$

$MustTerminate \triangleq$
$\Diamond(pstate \in \{\text{“terminated”}, \text{“exceptionThrown”}\})$

Initialization

$$
\begin{array}{l}
\mathit{Init} \triangleq \\
\quad \land \mathit{fstates} = [r \in \mathit{IDRange} \mapsto \\
\qquad [\mathit{id} \mapsto \mathit{NotID},\ \mathit{status} \mapsto \text{“unused”},\ \mathit{type} \mapsto \mathit{NotType}, \\
\qquad\ \mathit{count} \mapsto 0,\ \mathit{excs} \mapsto \langle\rangle,\ \mathit{here} \mapsto \mathit{NotPlace}, \\
\qquad\ \mathit{parent} \mapsto \mathit{NotID},\ \mathit{root} \mapsto \mathit{NotID},\ \mathit{isGlobal} \mapsto \text{FALSE}, \\
\qquad\ \mathit{remActs} \mapsto [p \in \mathit{PLACE} \mapsto 0]]] \\
\quad \land \mathit{readyQ} = [p \in \mathit{PLACE} \mapsto \langle\rangle] \\
\quad \land \mathit{msgs} = \{\} \\
\quad \land \mathit{pstate} = \text{“init”} \\
\quad \land \mathit{program} = [b \in \mathit{BlockID} \mapsto \\
\qquad [b \mapsto \mathit{NotBlockID},\ \mathit{type} \mapsto \text{“NA”},\ \mathit{dst} \mapsto \mathit{NotPlace}, \\
\qquad\ \mathit{mxstmt} \mapsto 0,\ \mathit{stmts} \mapsto [s \in \mathit{StmtID} \mapsto \mathit{NotBlockID}], \\
\qquad\ \mathit{ran} \mapsto \text{FALSE}]] \\
\quad \land \mathit{aseq} = 1 \\
\quad \land \mathit{fseq} = \mathit{FIRST\_ID} \\
\quad \land \mathit{mseq} = 0 \\
\quad \land \mathit{ppProgram} = \mathit{PROG} \\
\quad \land \mathit{ppcurStmt} = 0 \\
\quad \land \mathit{incPar} = [p \in \mathit{PLACE} \mapsto 0] \\
\quad \land \mathit{decPar} = [p \in \mathit{PLACE} \mapsto 0] \\
\quad \land \mathit{thrds} = [p \in \mathit{PLACE} \mapsto \\
\qquad\ [t \in \mathit{ThreadID} \mapsto \\
\qquad\ \text{IF } t < \mathit{NTHREADS} \\
\qquad\quad \text{THEN } [\mathit{tid} \mapsto t,\ \mathit{status} \mapsto \text{“idle”},\ \mathit{stack} \mapsto \langle\rangle] \\
\qquad\quad \text{ELSE } [\mathit{tid} \mapsto t,\ \mathit{status} \mapsto \text{“NA”},\ \mathit{stack} \mapsto \langle\rangle]]] \\
\quad \land \mathit{killed} = \{\} \\
\quad \land \mathit{killedCnt} = 0 \\
\quad \land \mathit{pendingAct} = \{\} \\
\quad \land \mathit{isDead} = [p \in \mathit{PLACE} \mapsto [z \in \mathit{PLACE} \mapsto \text{FALSE}]\ ] \\
\quad \land \mathit{p0fstates} = [r \in \mathit{IDRange} \mapsto [\ \mathit{id} \mapsto \mathit{NotID}, \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{parent} \mapsto \mathit{NotID}, \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{gfsRoot} \mapsto \mathit{NotID}, \\
\qquad\qquad\qquad\qquad\qquad \mathit{gfsRootPlace} \mapsto \mathit{NotPlace}, \\
\qquad\qquad\qquad\qquad\qquad\quad \mathit{numActive} \mapsto 0, \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \mathit{live} \mapsto [p \in \mathit{PLACE} \mapsto 0], \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{transit} \mapsto [p \in \mathit{PLACE} \mapsto [q \in \mathit{PLACE} \mapsto 0]], \\
\qquad\qquad\qquad\qquad\qquad\ \mathit{liveAdopted} \mapsto [p \in \mathit{PLACE} \mapsto 0], \\
\qquad\qquad\qquad\qquad\qquad \mathit{transitAdopted} \mapsto [p \in \mathit{PLACE} \mapsto [q \in \mathit{PLACE} \mapsto 0]], \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \mathit{excs} \mapsto \langle\rangle, \\
\qquad\qquad\qquad\qquad\qquad\quad \mathit{adopterId} \mapsto \mathit{NotID}, \\
\qquad\qquad\qquad\qquad\qquad\quad \mathit{isReleased} \mapsto \text{FALSE} \\
\qquad\qquad\qquad\qquad\qquad\qquad\ ] \\
\qquad\qquad\qquad\ ] \\
\quad \land \mathit{p0dead} = \mathit{NotPlace}
\end{array}
$$

$\land\ p0adoptSet = \{\}$
$\land\ p0convSet = \{\}$
$\land\ p0state =$ "running"

Parsing the input program into another format for easier processing

$ParseInputProgram \triangleq$
$\land\ pstate =$ "init"
$\land\ Len(ppProgram) > 0$
$\land$ LET $curBlk \triangleq Head(ppProgram)$
$body \triangleq curBlk.body$
$t \triangleq curBlk.type$
$d \triangleq curBlk.dst$
$b \triangleq curBlk.b$
$h \triangleq$ IF $body = \langle\rangle$ THEN $EMPTY\_BLOCK$ ELSE $Head(body)$
IN $\land\ program' = [program$ EXCEPT $![b].b = b,$
$![b].type = t,$
$![b].dst = d,$
$![b].mxstmt = ppcurStmt,$
$![b].ran =$ FALSE,
$![b].stmts[ppcurStmt] = h]$
$\land$ IF $((Len(body) = 0 \land ppcurStmt = 0) \lor Len(body) = 1)$
THEN $\land\ ppcurStmt' = 0$
$\land\ ppProgram' = Tail(ppProgram)$
ELSE $\land\ ppcurStmt' = ppcurStmt + 1$
$\land\ ppProgram' = \langle[type \mapsto t,$
$dst \mapsto d,$
$b \mapsto b,$
$body \mapsto Tail(body),$
$err \mapsto$ ""$]$
$\rangle \circ Tail(ppProgram)$
$\land$ UNCHANGED $\langle fstates, pstate, msgs, aseq, fseq, mseq, readyQ,$
$thrds, incPar, decPar, p0dead, p0adoptSet, p0state, p0convSet,$
$killed, killedCnt, p0fstates, pendingAct, isDead\rangle$

Start program execution (*i.e.* simulate $X10$'s main method)

$Run \triangleq$
$\land\ pstate =$ "init"
$\land\ Len(ppProgram) = 0$
$\land\ pstate' =$ "running"
$\land$ LET $curStmt \triangleq$ IF $program[0].type =$ "finish" THEN -2 ELSE -1
IN $thrds' = [thrds$ EXCEPT $![PROG\_HOME][0].stack =$
$\langle[\ b \mapsto 0,$
$i \mapsto curStmt,$
$fid \mapsto NoParent]$
$\rangle,$

$$! \quad [PROG\_HOME][0].status = \text{“running”}]$$
$\land\ program' = [program \text{ EXCEPT } ![0].ran = \text{TRUE}]$
$\land$ UNCHANGED $\langle fstates,\ msgs,\ aseq,\ fseq,\ mseq,\ readyQ,\ ppProgram,$
$ppcurStmt,\ incPar,\ decPar,\ p0dead,\ p0adoptSet,\ p0state,\ p0convSet,$
$killed,\ killedCnt,\ p0fstates,\ pendingAct,\ isDead\rangle$

Helper Actions

Push activity to the ready queue

$PushReadyFIFO(here,\ activity) \triangleq$
$\land\ readyQ' = [readyQ \text{ EXCEPT } ![here] = Append(@,\ activity)]$

poll activity from the ready queue

$PollReadyFIFO(here) \triangleq$
$\land\ readyQ[here] \neq \langle\rangle$
$\land\ readyQ' = [readyQ \text{ EXCEPT } ![here] = Tail(readyQ[here])]$

record activity pending approval from the resilient store

$AddPendingActivity(activity) \triangleq$
$\land\ pendingAct' = pendingAct \cup \{activity\}$

search for a pending activity given its *id*

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

Push an activity received from another place

$RecvAndSubmitRemoteActivity(here,\ src,\ act,\ msg) \triangleq$
$\land\ Finish(act.fid)!NotifyRemoteActivityCreation(src,\ act,\ msg)$
$\land$ IF $Finish(act.fid)!IsResilient$
THEN $\land\ AddPendingActivity(act)$
$\land\ readyQ' = readyQ$
ELSE $\land\ PushReadyFIFO(here,\ act)$
$\land\ pendingAct' = pendingAct$
$\land\ mseq' = mseq$

Push a local activity

$SubmitLocalActivity(here,\ act) \triangleq$
$\land$ IF $act.fid \neq NoParent$
THEN $Finish(act.fid)!NotifyLocalActivitySpawnAndCreation(here,\ act)$
ELSE $fstates' = fstates$
$\land\ PushReadyFIFO(here,\ act)$

Increase/decrease parallelism actions

Increase the number of worker threads

$IncreaseParallelism(here) \triangleq$
$\land\ here \notin killed$
$\land\ pstate = \text{“running”}$
$\land\ incPar[here] > 0$
$\land$ LET $tid \triangleq FindThread(here, \text{“NA”})$
IN $\land\ tid \neq NotThreadID$
$\land\ incPar' = [incPar \text{ EXCEPT } ![here] = @ - 1]$
$\land\ thrds' = [thrds \text{ EXCEPT } ![here][tid].status = \text{“idle”}]$
$\land$ UNCHANGED $\langle fstates, msgs, pstate, program, aseq, fseq, mseq, readyQ, ppProgram, ppcurStmt, decPar, p0dead, p0adoptSet, p0state, p0convSet, killed, killedCnt, p0fstates, pendingAct, isDead\rangle$

Decrease the number of worker threads

$DecreaseParallelism(here) \triangleq$
$\land\ here \notin killed$
$\land\ pstate = \text{“running”}$
$\land\ decPar[here] > 0$
$\land$ LET $tid \triangleq FindThread(here, \text{“idle”})$
IN $\land\ tid \neq NotThreadID$
$\land\ decPar' = [decPar \text{ EXCEPT } ![here] = @ - 1]$
$\land\ thrds' = [thrds \text{ EXCEPT } ![here][tid].status = \text{“NA”}]$
$\land$ UNCHANGED $\langle fstates, msgs, pstate, program, aseq, fseq, mseq, readyQ, ppProgram, ppcurStmt, incPar, p0dead, p0adoptSet, p0state, p0convSet, killed, killedCnt, p0fstates, pendingAct, isDead\rangle$

Program Execution Actions

Idle thread fetching a ready activity

$IThreadFetchActivity(here, tid) \triangleq$
$\land\ here \notin killed$
$\land\ pstate = \text{“running”}$
$\land\ thrds[here][tid].status = \text{“idle”}$
$\land\ PollReadyFIFO(here)$
$\land$ LET $act \triangleq Head(readyQ[here])$
$stkEntry \triangleq [b \mapsto act.b,\ i \mapsto -1,\ fid \mapsto act.fid]$
IN $\land\ thrds' = [thrds \text{ EXCEPT } ![here][tid].stack = \langle stkEntry\rangle, ![here][tid].status = \text{“running”}]$
$\land\ program' = [program \text{ EXCEPT } ![act.b].ran = \text{TRUE}]$
$\land$ UNCHANGED $\langle fstates, msgs, pstate, aseq, fseq, mseq, ppProgram, ppcurStmt, incPar, decPar, p0dead, p0adoptSet, p0state, p0convSet, killed, killedCnt, p0fstates, pendingAct, isDead\rangle$

Running thread processing an expression

$RThreadRunExpr(here, tid) \triangleq$
$\land\ here \notin killed$

$\land\ pstate = \text{“running”}$
$\land\ thrds[here][tid].status = \text{“running”}$
$\land$ LET $top \triangleq Head(thrds[here][tid].stack)$
$blk \triangleq top.b$
IN $\land\ program[blk].type = \text{“expr”}$
$\land$ IF $Len(thrds[here][tid].stack) = 1$
THEN $thrds' = [thrds$ EXCEPT $![here][tid].stack = \langle\rangle,$
$![here][tid].status = \text{“idle”}]$
ELSE $thrds' = [thrds$ EXCEPT $![here][tid].stack = Tail(@)]$
$\land$ UNCHANGED $\langle fstates, msgs, pstate, program, aseq, fseq, mseq, readyQ,$
$ppProgram, ppcurStmt, incPar, decPar, p0dead, p0adoptSet, p0state, p0convSet,$
$killed, killedCnt, p0fstates, pendingAct, isDead\rangle$

Running thread processing a kill statement

$RThreadRunKill(here, tid) \triangleq$
$\land\ here \notin killed$
$\land\ here \neq PROG\_HOME$
$\land\ pstate = \text{“running”}$
$\land\ thrds[here][tid].status = \text{“running”}$
$\land$ LET $top \triangleq Head(thrds[here][tid].stack)$
$blk \triangleq top.b$
IN $\land\ program[blk].type = \text{“kill”}$
$\land\ killed' = killed \cup \{here\}$
$\land\ killedCnt' = killedCnt + 1$
$\land$ UNCHANGED $\langle fstates, msgs, pstate, program, aseq, fseq, mseq, p0dead,$
$readyQ, thrds, ppProgram, ppcurStmt, incPar, decPar, p0convSet,$
$p0fstates, pendingAct, isDead, p0adoptSet, p0state\rangle$

Running thread processing the end of an *async* block

$RThreadRunAsyncEnd(here, tid) \triangleq$
$\land\ here \notin killed$
$\land\ pstate = \text{“running”}$
$\land\ thrds[here][tid].status = \text{“running”}$
$\land$ LET $top \triangleq Head(thrds[here][tid].stack)$
$blk \triangleq top.b$
$fid \triangleq top.fid$
IN $\land\ program[blk].type = \text{“async”}$
$\land\ program[blk].mxstmt = top.i$
$\land\ Finish(fid)!NotifyActivityTermination$
$\land$ IF $Len(thrds[here][tid].stack) = 1$
THEN $thrds' = [thrds$ EXCEPT $![here][tid].stack = \langle\rangle,$
$![here][tid].status = \text{“idle”}]$
ELSE $thrds' = [thrds$ EXCEPT $![here][tid].stack = Tail(@)]$
$\land$ IF $blk = 0$
THEN $pstate' = \text{“terminated”}$

```
                ELSE  pstate' = pstate
    ∧ UNCHANGED ⟨msgs, pstate, program, aseq, fseq, mseq, readyQ, ppProgram,
                 ppcurStmt, incPar, decPar, p0dead, p0adoptSet, p0state, p0convSet,
                 killed, killedCnt, p0fstates, pendingAct, isDead⟩

 Running thread processing the end of a finish block and blocking itself
RThreadRunFinishEnd(here, tid) ≜
    ∧ here ∉ killed
    ∧ pstate = "running"
    ∧ thrds[here][tid].status = "running"
    ∧ LET top ≜ Head(thrds[here][tid].stack)
      IN  ∧ program[top.b].type = "finish"
          ∧ program[top.b].mxstmt = top.i
          ∧ Finish(top.fid)!NotifyActivityTermination
          ∧ thrds' = [thrds EXCEPT ![here][tid].status = "blockedFinish"]
          ∧ incPar' = [incPar EXCEPT ![here] = @ + 1]
    ∧ UNCHANGED ⟨msgs, pstate, program, aseq, fseq, mseq, readyQ, ppProgram,
                 ppcurStmt, decPar, p0dead, p0adoptSet, p0state, p0convSet,
                 killed, killedCnt, p0fstates, pendingAct, isDead⟩

 Terminated finish unblocks its thread
BThreadUnblockFinish(here, tid) ≜
    ∧ here ∉ killed
    ∧ pstate = "running"
    ∧ thrds[here][tid].status = "blockedFinish"
    ∧ LET top ≜ Head(thrds[here][tid].stack)
          blk ≜ top.b
          fid ≜ top.fid
      IN  ∧ program[blk].type = "finish"
          ∧ program[blk].mxstmt = top.i
          ∧ Finish(fid)!Terminated
          ∧ decPar' = [decPar EXCEPT ![here] = @ + 1]
          ∧ IF   Len(thrds[here][tid].stack) = 1
            THEN thrds' = [thrds EXCEPT ![here][tid].stack = ⟨⟩,
                                        ![here][tid].status = "idle"]
            ELSE thrds' = [thrds EXCEPT ![here][tid].stack = Tail(@),
                                        ![here][tid].status = "running"]
          ∧ IF   blk = 0 ∧ Finish(fid)!HasExceptions
            THEN pstate' = "exceptionThrown"
            ELSE IF blk = 0 ∧ ¬Finish(fid)!HasExceptions
            THEN pstate' = "terminated"
            ELSE pstate' = pstate
    ∧ UNCHANGED ⟨fstates, msgs, program, aseq, fseq, mseq, readyQ,
                 ppProgram, ppcurStmt, incPar, p0dead, p0adoptSet, p0state, p0convSet,
                 killed, killedCnt, p0fstates, pendingAct, isDead⟩
```

Running thread processing the beginning of a finish block

$RThreadRunFinishFirstStmt(here,\ tid) \triangleq$
$\quad \land\ here \notin killed$
$\quad \land\ pstate = \text{“running”}$
$\quad \land\ thrds[here][tid].status = \text{“running”}$
$\quad \land$ LET $top \triangleq Head(thrds[here][tid].stack)$
$\qquad\quad tail \triangleq Tail(thrds[here][tid].stack)$
$\qquad\quad lstStmt \triangleq top.i$
$\qquad\quad curStmt \triangleq top.i + 1$
$\qquad\quad blk \triangleq top.b$
$\qquad\quad fid \triangleq top.fid$
$\qquad$ IN $\land\ program[blk].type = \text{“finish”}$
$\qquad\qquad \land\ lstStmt = -2$
$\qquad\qquad \land\ Finish(fseq)!Alloc(ROOT\_FINISH,\ here,\ fid,\ fseq)$
$\qquad\qquad \land\ thrds' = [thrds$ EXCEPT $![here][tid].stack = \langle [b \mapsto top.b,$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad i \mapsto curStmt,$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad fid \mapsto fseq]$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \rangle \circ tail]$
$\qquad\qquad \land\ fseq' = fseq + 1$
$\quad \land$ UNCHANGED $\langle msgs,\ pstate,\ program,\ aseq,\ mseq,\ readyQ,\ ppProgram,$
$\qquad\qquad\qquad ppcurStmt,\ incPar,\ decPar,\ p0dead,\ p0adoptSet,\ p0state,\ p0convSet,$
$\qquad\qquad\qquad killed,\ killedCnt,\ p0fstates,\ pendingAct,\ isDead\rangle$

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

$RThreadRunNestedLocalAsync(here,\ tid) \triangleq$
$\quad \land\ here \notin killed$
$\quad \land\ pstate = \text{“running”}$
$\quad \land\ thrds[here][tid].status = \text{“running”}$
$\quad \land$ LET $top \triangleq Head(thrds[here][tid].stack)$
$\qquad\quad tail \triangleq Tail(thrds[here][tid].stack)$
$\qquad\quad lstStmt \triangleq top.i$
$\qquad\quad curStmt \triangleq top.i + 1$
$\qquad\quad blk \triangleq top.b$
$\qquad\quad fid \triangleq top.fid$
$\qquad\quad nested \triangleq program[blk].stmts[curStmt]$
$\qquad$ IN $\land\ program[blk].type \notin \{\text{“expr”},\ \text{“kill”}\}$
$\qquad\qquad \land\ curStmt \geq 0$
$\qquad\qquad \land\ curStmt \leq program[blk].mxstmt$
$\qquad\qquad \land\ program[nested].type = \text{“async”}$
$\qquad\qquad \land\ program[nested].dst = here$
$\qquad\qquad \land\ SubmitLocalActivity(here,\ [aid \mapsto aseq,$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad b \mapsto nested,$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad fid \mapsto fid])$
$\qquad\qquad \land\ aseq' = aseq + 1$
$\qquad\qquad \land\ thrds' = [thrds$ EXCEPT $![here][tid].stack =$

$$\langle [\ b \mapsto top.b,\ i \mapsto curStmt,\ fid \mapsto fid] \rangle \circ tail]$$

$\wedge$ UNCHANGED $\langle msgs, pstate, program, fseq, mseq, ppProgram, ppcurStmt, incPar, decPar, p0dead, p0adoptSet, p0state, p0convSet, killed, killedCnt, p0fstates, pendingAct, isDead\rangle$

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

$RThreadRunNestedRemoteAsync(here, tid) \triangleq$
- $\wedge\ here \notin killed$
- $\wedge\ pstate = \text{“running”}$
- $\wedge\ thrds[here][tid].status = \text{“running”}$
- $\wedge$ LET $top \triangleq Head(thrds[here][tid].stack)$
  - $tail \triangleq Tail(thrds[here][tid].stack)$
  - $lstStmt \triangleq top.i$
  - $curStmt \triangleq top.i + 1$
  - $blk \triangleq top.b$
  - $fid \triangleq top.fid$
  - $root \triangleq GetRootFinishId(fid)$
  - $nested \triangleq program[blk].stmts[curStmt]$
- IN
  - $\wedge\ program[blk].type \notin \{\text{“expr”}, \text{“kill”}\}$
  - $\wedge\ fid \neq NoParent$
  - $\wedge\ curStmt \geq 0$
  - $\wedge\ curStmt \leq program[blk].mxstmt$
  - $\wedge\ program[nested].type = \text{“async”}$
  - $\wedge\ program[nested].dst \neq here$
  - $\wedge$ IF $Finish(fid)!IsResilient$
    - THEN $\wedge\ Finish(fid)!NotifySubActivitySpawn(program[nested].dst)$
      - $\wedge\ thrds' = [thrds$ EXCEPT $![here][tid].status = \text{“blockedAsync”}]$
      - $\wedge\ incPar' = [incPar$ EXCEPT $![here] = @ + 1]$
    - ELSE $\vee\ \wedge\ Finish(fid)!NotifySubActivitySpawn(program[nested].dst)$
      - $\wedge\ SendMsg([mid \mapsto mseq, src \mapsto here, dst \mapsto program[nested].dst, type \mapsto \text{“async”}, fid \mapsto root, b \mapsto nested])$
      - $\wedge\ mseq' = mseq + 1$
      - $\wedge\ thrds' = [thrds$ EXCEPT $![here][tid].stack = \langle [\ b \mapsto top.b,\ i \mapsto curStmt,\ fid \mapsto fid] \rangle \circ tail]$
      - $\wedge\ incPar' = incPar$

$$
\begin{array}{l}
\quad \lor \ \land \ Finish(fid)!NotifySubActivitySpawnError(program[nested].dst) \\
\qquad \land \ msgs' = msgs \\
\qquad \land \ mseq' = mseq \\
\qquad \land \ thrds' = [thrds \ \text{EXCEPT} \ ![here][tid].stack = \\
\qquad\qquad\qquad\qquad \langle [b \ \mapsto top.b, \\
\qquad\qquad\qquad\qquad\ \ i \ \mapsto program[blk].mxstmt, \\
\qquad\qquad\qquad\qquad\ fid \mapsto fid] \\
\qquad\qquad\qquad\qquad \rangle \circ tail] \\
\qquad \land \ incPar' = incPar \\
\land \ \text{UNCHANGED} \ \langle pstate, \ program, \ aseq, \ fseq, \ readyQ, \ ppProgram, \\
\qquad\qquad\qquad ppcurStmt, \ decPar, \ p0dead, \ p0adoptSet, \ p0state, \ p0convSet, \\
\qquad\qquad\qquad killed, \ killedCnt, \ p0fstates, \ pendingAct, \ isDead \rangle
\end{array}
$$

Processing a nested finish in the currently running block

$$
\begin{array}{l}
RThreadRunNestedFinish(here, \ tid) \ \triangleq \\
\land \ here \notin killed \\
\land \ pstate = \text{“running”} \\
\land \ thrds[here][tid].status = \text{“running”} \\
\land \ \text{LET} \ top \ \triangleq \ Head(thrds[here][tid].stack) \\
\qquad\quad tail \ \triangleq \ Tail(thrds[here][tid].stack) \\
\qquad\quad lstStmt \ \triangleq \ top.i \\
\qquad\quad curStmt \ \triangleq \ top.i + 1 \\
\qquad\quad blk \ \triangleq \ top.b \\
\qquad\quad fid \ \triangleq \ top.fid \\
\qquad\ nested \ \triangleq \ program[blk].stmts[curStmt] \\
\quad \text{IN} \quad \land \ program[blk].type \notin \{\text{“expr”}, \ \text{“kill”}\} \\
\qquad\quad \land \ curStmt \geq 0 \\
\qquad\quad \land \ curStmt \leq program[blk].mxstmt \\
\qquad\quad \land \ program[nested].type = \text{“finish”} \\
\qquad\quad \land \ program[nested].dst = here \\
\qquad\quad \land \ thrds' = [thrds \ \text{EXCEPT} \ ![here][tid].stack = \\
\qquad\qquad\qquad\qquad\qquad\qquad \langle [\ \ b \ \mapsto nested, \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \ i \ \mapsto -2, \\
\qquad\qquad\qquad\qquad\qquad\qquad\ fid \mapsto fid], \\
\qquad\qquad\qquad\qquad\qquad\qquad [\ \ b \ \mapsto top.b, \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \ i \ \mapsto curStmt, \\
\qquad\qquad\qquad\qquad\qquad\qquad\ fid \mapsto fid] \\
\qquad\qquad\qquad\qquad\qquad\qquad \rangle \circ tail] \\
\qquad\quad \land \ program' = [program \ \text{EXCEPT} \ ![nested].ran = \text{TRUE}] \\
\land \ \text{UNCHANGED} \ \langle fstates, \ msgs, \ pstate, \ aseq, \ fseq, \ mseq, \ readyQ, \\
\qquad\qquad\qquad ppProgram, \ ppcurStmt, \ incPar, \ decPar, \ p0dead, \ p0adoptSet, \ p0state, \ p0convSet, \\
\qquad\qquad\qquad killed, \ killedCnt, \ p0fstates, \ pendingAct, \ isDead \rangle
\end{array}
$$

Processing a nested expression in the currently running block

$$
RThreadRunNestedExprORKill(here, \ tid) \ \triangleq
$$

$\land\ here \notin killed$
$\land\ pstate = \text{“running”}$
$\land\ thrds[here][tid].status = \text{“running”}$
$\land$ LET $top \triangleq Head(thrds[here][tid].stack)$
$tail \triangleq Tail(thrds[here][tid].stack)$
$lstStmt \triangleq top.i$
$curStmt \triangleq top.i + 1$
$blk \triangleq top.b$
$fid \triangleq top.fid$
$nested \triangleq program[blk].stmts[curStmt]$
IN $\land\ program[blk].type \notin \{\text{“expr”}, \text{“kill”}\}$
$\land\ curStmt \geq 0$
$\land\ curStmt \leq program[blk].mxstmt$
$\land\ program[nested].type \in \{\text{“expr”}, \text{“kill”}\}$
$\land\ program[nested].dst = here$
$\land\ thrds' = [thrds$ EXCEPT $![here][tid].stack =$
$\langle[\ b \mapsto nested,$
$i \mapsto -1,$
$fid \mapsto fid],$
$[\ b \mapsto top.b,$
$i \mapsto curStmt,$
$fid \mapsto fid]$
$\rangle \circ tail]$
$\land\ program' = [program$ EXCEPT $![nested].ran =$ TRUE$]$
$\land$ UNCHANGED $\langle fstates, msgs, pstate, aseq, fseq, mseq, readyQ,$
$ppProgram, ppcurStmt, incPar, decPar, p0dead, p0adoptSet, p0state, p0convSet,$
$killed, killedCnt, p0fstates, pendingAct, isDead\rangle$

Processing a nested error in the currently running block

$RThreadRunNestedError(here, tid) \triangleq$
$\land\ here \notin killed$
$\land\ pstate = \text{“running”}$
$\land\ thrds[here][tid].status = \text{“running”}$
$\land$ LET $top \triangleq Head(thrds[here][tid].stack)$
$tail \triangleq Tail(thrds[here][tid].stack)$
$lstStmt \triangleq top.i$
$curStmt \triangleq top.i + 1$
$blk \triangleq top.b$
$fid \triangleq top.fid$
$nested \triangleq program[blk].stmts[curStmt]$
IN $\land\ program[blk].type \notin \{\text{“expr”}, \text{“error”}, \text{“kill”}\}$
$\land\ curStmt \geq 0$
$\land\ curStmt \leq program[blk].mxstmt$
$\land\ program[nested].type = \text{“error”}$
$\land\ program[nested].dst = here$

$\land\ thrds' = [thrds \text{ EXCEPT } ![here][tid].stack =$ jump to the end of the current block
$\langle[\ b \mapsto top.b,$
$\ \ i \mapsto program[blk].mxstmt,$
$\ \ fid \mapsto fid]$
$\rangle \circ tail]$
$\land\ program' = [program \text{ EXCEPT } ![nested].ran = \text{TRUE}]$
$\land\ Finish(fid)!PushException([err \mapsto \text{“ErrorStmt”}, from \mapsto here])$
$\land$ UNCHANGED $\langle msgs, pstate, aseq, fseq, mseq, readyQ, p0adoptSet, p0state, p0convSet,$
$ppProgram, ppcurStmt, incPar, decPar, p0dead,$
$killed, killedCnt, p0fstates, pendingAct, isDead\rangle$

Parsing an incoming *async* and creating its *RemoteFinish* object

$CreateRemoteFinish(here) \triangleq$
$\land\ here \notin killed$
$\land\ pstate = \text{“running”}$
$\land$ LET $msg \triangleq FindIncomingMSG(here, \text{“async”})$
$pid \triangleq msg.fid$
$fid \triangleq GetActiveFID(REMOTE\_FINISH, here, pid)$
IN $\land\ pid \neq NotID$
$\land\ fid = NotID$
$\land\ Finish(fseq)!Alloc(REMOTE\_FINISH, here, pid, pid)$
$\land\ fseq' = fseq + 1$
$\land$ UNCHANGED $\langle msgs, pstate, program, aseq, mseq, readyQ, thrds, ppProgram,$
$ppcurStmt, incPar, decPar, p0dead, p0adoptSet, p0state, p0convSet,$
$killed, killedCnt, p0fstates, pendingAct, isDead\rangle$

Pushing an incoming *async* to the ready queue

$RecvAsync(here) \triangleq$
$\land\ here \notin killed$
$\land\ pstate = \text{“running”}$
$\land$ LET $msg \triangleq FindIncomingMSG(here, \text{“async”})$
$pid \triangleq msg.fid$
$fid \triangleq GetActiveFID(REMOTE\_FINISH, here, pid)$
$src \triangleq msg.src$
$blk \triangleq msg.b$
IN $\land\ pid \neq NotID$
$\land\ fid \neq NotID$
$\land\ src \neq NotPlace$
$\land\ RecvAndSubmitRemoteActivity(here, src, [aid \mapsto aseq, b \mapsto blk, fid \mapsto fid],$
$[\ mid \mapsto msg.mid,$
$\ \ src \mapsto msg.src,$
$\ \ dst \mapsto here,$
$\ \ type \mapsto \text{“async”},$
$\ \ b \mapsto blk,$

$fid \mapsto pid])$

$\land\ aseq' = aseq + 1$

$\land$ UNCHANGED $\langle pstate, program, fseq, thrds, ppProgram, ppcurStmt,$
$p0dead, p0adoptSet, p0state, p0convSet,$
$incPar, decPar, killed, killedCnt, p0fstates, isDead\rangle$

Enclosing finish receiving a termination signal from a remote task

$RecvAsyncTerm(here) \triangleq$

$\land\ here \notin killed$

$\land\ pstate = \text{“running”}$

$\land$ LET $msg \triangleq FindIncomingMSG(here, \text{“asyncTerm”})$
$fid \triangleq msg.fid$
$src \triangleq msg.src$
IN $\land\ fid \neq NotID$
$\land\ src \neq NotPlace$
$\land\ Finish(fid)!ProcessChildTermMsg(msg)$

$\land$ UNCHANGED $\langle pstate, program, aseq, fseq, mseq, readyQ, thrds, ppProgram,$
$ppcurStmt, incPar, decPar, p0dead, p0adoptSet, p0state, p0convSet,$
$killed, killedCnt, p0fstates, pendingAct, isDead\rangle$

RemoteFinish notifying its *RootFinish* that it terminated

$NotifyParentFinish(fid) \triangleq$

$\land\ fstates[fid].here \notin killed$

$\land\ pstate = \text{“running”}$

$\land\ fstates[fid].status = \text{“finished”}$

$\land$ LET $type \triangleq fstates[fid].type$
$pid \triangleq fstates[fid].root$
IN $\land\ Finish(fid)!SendTermMsg$

$\land$ UNCHANGED $\langle program, pstate, aseq, fseq, readyQ, thrds, ppProgram,$
$ppcurStmt, incPar, decPar, p0dead, p0adoptSet, p0state, p0convSet,$
$killed, killedCnt, p0fstates, pendingAct, isDead\rangle$

Simulating place failure

$Kill(here) \triangleq$

$\land\ pstate = \text{“running”}$

$\land\ here \neq PROG\_HOME$

$\land\ here \notin killed$

$\land\ killedCnt < MXDEAD$

$\land\ killed' = killed \cup \{here\}$

$\land\ killedCnt' = killedCnt + 1$

$\land$ UNCHANGED $\langle fstates, msgs, pstate, program, aseq, fseq, mseq,$
$readyQ, thrds, ppProgram, ppcurStmt, incPar, decPar, p0adoptSet, p0state, p0convSet,$
$p0fstates, pendingAct, isDead, p0dead\rangle$

When a place detects that another place has died for the first time

$NotifyPlaceDeath(here) \triangleq$
$\land\ here \notin killed$
$\land\ pstate = \text{“running”}$
$\land$ LET $allNewDead \triangleq \{p \in killed : \neg isDead[here][p]\}$
$oneNewDead \triangleq$ IF $allNewDead = \{\}$ THEN $NotPlace$ ELSE CHOOSE $p \in allNewDead$: TRUE
IN $\land\ oneNewDead \neq NotPlace$
$\land\ isDead[here][oneNewDead] =$ FALSE
$\land\ isDead' = [isDead$ EXCEPT $![here][oneNewDead] =$ TRUE$]$
$\land$ IF $here = PROG\_HOME$
THEN $P0NotifyPlaceDeath(oneNewDead)$
ELSE $p0dead' = p0dead \land p0adoptSet' = p0adoptSet \land p0state' = p0state$
$\land$ UNCHANGED $\langle fstates, msgs, pstate, program, aseq, fseq, mseq,$
$readyQ, thrds, ppProgram, ppcurStmt, incPar, decPar, p0convSet,$
$killed, killedCnt, p0fstates, pendingAct\rangle$

resilient store has changed the transit counters
$BThreadUnblockResilientAsync(here, tid) \triangleq$
$\land\ here \notin killed$
$\land\ pstate = \text{“running”}$
$\land\ thrds[here][tid].status = \text{“blockedAsync”}$
$\land\ msgs \neq \{\}$
$\land$ LET $msg \triangleq FindIncomingMSG(here, \text{“transitDone”})$
$top \triangleq Head(thrds[here][tid].stack)$
$tail \triangleq Tail(thrds[here][tid].stack)$
$lstStmt \triangleq top.i$
$curStmt \triangleq top.i + 1$
$blk \triangleq top.b$
$fid \triangleq top.fid$
$root \triangleq GetRootFinishId(fid)$
$nested \triangleq program[blk].stmts[curStmt]$
IN $\land\ msg \neq NotMessage$
$\land\ decPar' = [decPar$ EXCEPT $![here] = @ + 1]$
$\land\ ReplaceMsg([mid \mapsto msg.mid,$
$src \mapsto PROG\_HOME,$
$dst \mapsto here,$
$fid \mapsto msg.fid,$
$type \mapsto \text{“transitDone”}],$
$[mid \mapsto mseq,$
$src \mapsto here,$
$dst \mapsto program[nested].dst,$
$type \mapsto \text{“async”},$
$fid \mapsto root,$
$b \mapsto nested])$
$\land\ mseq' = mseq + 1$
$\land\ thrds' = [thrds$ EXCEPT $![here][tid].status = \text{“running”},$

```
                                        ![here][tid].stack =
                                                ⟨[ b   ↦ top.b,
                                                   i   ↦ curStmt,
                                                   fid ↦ fid]
                                                ⟩ ∘ tail]
    ∧ UNCHANGED ⟨fstates, pstate, program, aseq, fseq,
          readyQ, ppProgram, ppcurStmt, incPar, p0dead, p0adoptSet, p0state, p0convSet,
          killed, killedCnt, p0fstates, pendingAct, isDead⟩

 resilient store has given permission to execute a remote activity
SubmitPendingActivity(here) ≜
    ∧ here ∉ killed
    ∧ pstate = "running"
    ∧ pendingAct ≠ {}
    ∧ msgs ≠ {}
    ∧ LET msg ≜ FindIncomingMSG(here, "liveDone")
          actId ≜ msg.aid
          activity ≜ FindPendingActivity(actId)
          submit ≜ msg.submit
      IN  ∧ msg ≠ NotMessage
          ∧ activity ≠ NotActivity
          ∧ RecvMsg([ mid ↦ msg.mid,
                      src ↦ PROG_HOME,
                      dst ↦ here,
                      aid ↦ actId,
                      submit ↦ submit,
                      type ↦ "liveDone"])
          ∧ IF submit
              THEN PushReadyFIFO(here, activity)
              ELSE readyQ' = readyQ
          ∧ pendingAct' = pendingAct \ {activity}
    ∧ UNCHANGED ⟨fstates, pstate, program, aseq, fseq, mseq,
          thrds, ppProgram, ppcurStmt, incPar, decPar, p0dead, p0adoptSet, p0state, p0convSet,
          killed, killedCnt, p0fstates, isDead⟩

ReleaseRootFinish(here) ≜
    ∧ here ∉ killed
    ∧ pstate = "running"
    ∧ msgs ≠ {}
    ∧ LET msg ≜ FindIncomingMSG(here, "releaseFinish")
          fid ≜ msg.fid
      IN  ∧ msg ≠ NotMessage
          ∧ RecvMsg([mid ↦ msg.mid,
                     src ↦ PROG_HOME,
                     dst ↦ here,
```

$fid \mapsto fid,$
$type \mapsto$ "releaseFinish"])
$\land fstates' = [fstates \text{ EXCEPT } ![fid].status =$ "forgotten"]
$\land$ UNCHANGED $\langle pstate, program, aseq, fseq, mseq, p0dead,$
$readyQ, thrds, ppProgram, ppcurStmt, incPar, decPar, p0adoptSet, p0state, p0convSet,$
$killed, killedCnt, p0fstates, pendingAct, isDead\rangle$

Predicate enumerating all possible next actions

$Next \triangleq$
- $\lor ParseInputProgram$
- $\lor Run$
- $\lor RecvTransit(PROG\_HOME)$ resilient finish
- $\lor RecvLive(PROG\_HOME)$ resilient finish
- $\lor RecvCompleted(PROG\_HOME)$ resilient finish
- $\lor SeekAdoption(PROG\_HOME)$ resilient finish
- $\lor ConvertDeadActivities(PROG\_HOME)$ resilient finish
- $\lor ReleaseAll(PROG\_HOME)$
- $\lor \exists here \in PLACE :$
  - $\lor IncreaseParallelism(here)$
  - $\lor DecreaseParallelism(here)$
  - $\lor CreateRemoteFinish(here)$
  - $\lor RecvAsync(here)$
  - $\lor RecvAsyncTerm(here)$
  - $\lor NotifyPlaceDeath(here)$ resilient finish
  - $\lor SubmitPendingActivity(here)$ resilient finish
  - $\lor ReleaseRootFinish(here)$ resilient finish
  - $\lor \exists tid \in ThreadID :$
    - $\lor IThreadFetchActivity(here, tid)$
    - $\lor RThreadRunExpr(here, tid)$
    - $\lor RThreadRunKill(here, tid)$
    - $\lor RThreadRunAsyncEnd(here, tid)$
    - $\lor RThreadRunFinishEnd(here, tid)$
    - $\lor BThreadUnblockFinish(here, tid)$
    - $\lor RThreadRunFinishFirstStmt(here, tid)$
    - $\lor RThreadRunNestedLocalAsync(here, tid)$
    - $\lor RThreadRunNestedRemoteAsync(here, tid)$
    - $\lor RThreadRunNestedFinish(here, tid)$
    - $\lor RThreadRunNestedExprORKill(here, tid)$
    - $\lor RThreadRunNestedError(here, tid)$
    - $\lor BThreadUnblockResilientAsync(here, tid)$ resilient finish
  - $\lor \exists fid \in IDRange :$
    - $\lor NotifyParentFinish(fid)$

Asserting fairness properties to all actions

$$
\begin{aligned}
&\mathit{Liveness} \triangleq \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{ParseInputProgram}) \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{Run}) \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{RecvTransit}(\mathit{PROG\_HOME})) && \text{resilient finish} \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{RecvLive}(\mathit{PROG\_HOME})) && \text{resilient finish} \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{RecvCompleted}(\mathit{PROG\_HOME})) && \text{resilient finish} \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{SeekAdoption}(\mathit{PROG\_HOME})) && \text{resilient finish} \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{ConvertDeadActivities}(\mathit{PROG\_HOME})) && \text{resilient finish} \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{ReleaseAll}(\mathit{PROG\_HOME})) \\
&\quad \land \forall \mathit{here} \in \mathit{PLACE} : \\
&\qquad \mathrm{WF}_{\mathit{Vars}}(\mathit{IncreaseParallelism}(\mathit{here})) \\
&\qquad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{DecreaseParallelism}(\mathit{here})) \\
&\qquad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{CreateRemoteFinish}(\mathit{here})) \\
&\qquad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{RecvAsync}(\mathit{here})) \\
&\qquad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{RecvAsyncTerm}(\mathit{here})) \\
&\qquad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{NotifyPlaceDeath}(\mathit{here})) && \text{resilient finish} \\
&\qquad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{SubmitPendingActivity}(\mathit{here})) && \text{resilient finish} \\
&\qquad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{ReleaseRootFinish}(\mathit{here})) && \text{resilient finish} \\
&\qquad \land \forall \mathit{tid} \in \mathit{ThreadID} : \\
&\qquad\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{IThreadFetchActivity}(\mathit{here}, \mathit{tid})) \\
&\qquad\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{RThreadRunExpr}(\mathit{here}, \mathit{tid})) \\
&\qquad\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{RThreadRunKill}(\mathit{here}, \mathit{tid})) \\
&\qquad\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{RThreadRunAsyncEnd}(\mathit{here}, \mathit{tid})) \\
&\qquad\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{RThreadRunFinishEnd}(\mathit{here}, \mathit{tid})) \\
&\qquad\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{BThreadUnblockFinish}(\mathit{here}, \mathit{tid})) \\
&\qquad\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{RThreadRunFinishFirstStmt}(\mathit{here}, \mathit{tid})) \\
&\qquad\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{RThreadRunNestedLocalAsync}(\mathit{here}, \mathit{tid})) \\
&\qquad\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{RThreadRunNestedRemoteAsync}(\mathit{here}, \mathit{tid})) \\
&\qquad\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{RThreadRunNestedFinish}(\mathit{here}, \mathit{tid})) \\
&\qquad\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{RThreadRunNestedExprORKill}(\mathit{here}, \mathit{tid})) \\
&\qquad\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{RThreadRunNestedError}(\mathit{here}, \mathit{tid})) \\
&\qquad\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{BThreadUnblockResilientAsync}(\mathit{here}, \mathit{tid})) && \text{resilient finish} \\
&\quad \land \forall \mathit{fid} \in \mathit{IDRange} : \\
&\qquad \mathrm{WF}_{\mathit{Vars}}(\mathit{NotifyParentFinish}(\mathit{fid}))
\end{aligned}
$$

Specification

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

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

*** Example programs

*PROG*1:

0 finish {

1 *async*{

```
2       expr;
    }
3 async at(p1){
4       expr;
    }
}
```

$NBLOCKS = 5$
$MXFINISHES = 2$
$PLACE = \{p0,\ p1\}$
$\langle$
$[b \mapsto 0,\ type \mapsto$ "finish", $dst \mapsto p0,\ body \mapsto \langle 1,\ 2\rangle]$,
$[b \mapsto 1,\ type \mapsto$ "async", $dst \mapsto p0,\ body \mapsto \langle 3\rangle]$,
$[b \mapsto 2,\ type \mapsto$ "async", $dst \mapsto p1,\ body \mapsto \langle 4\rangle]$,
$[b \mapsto 3,\ type \mapsto$ "expr", $dst \mapsto p0,\ body \mapsto \langle\rangle]$,
$[b \mapsto 4,\ type \mapsto$ "expr", $dst \mapsto p1,\ body \mapsto \langle\rangle]$
$\rangle$

$PROG2$:

```
0 finish {
1 async at(p1){
2       finish{
3         async at(p2){
4           expr;
          }
5         kill;
        }
    }
  }
```

$NBLOCKS = 6$
$MXFINISHES = 4$
$MUST\_NOT\_RUN = \{\}$
$PLACE = \{p0,\ p1,\ p2\}$
$\langle$
$[b \mapsto 0,\ type \mapsto$ "finish", $dst \mapsto p0,\ body \mapsto \langle 1\rangle]$,
$[b \mapsto 1,\ type \mapsto$ "async", $dst \mapsto p1,\ body \mapsto \langle 2\rangle]$,
$[b \mapsto 2,\ type \mapsto$ "finish", $dst \mapsto p1,\ body \mapsto \langle 3,\ 5\rangle]$,
$[b \mapsto 3,\ type \mapsto$ "async", $dst \mapsto p2,\ body \mapsto \langle 4\rangle]$,
$[b \mapsto 4,\ type \mapsto$ "expr", $dst \mapsto p2,\ body \mapsto \langle\rangle]$,
$[b \mapsto 5,\ type \mapsto$ "kill", $dst \mapsto p1,\ body \mapsto \langle\rangle]$
$\rangle$

```
**
```

* Modification History
* Last modified *Mon Nov* 06 19:04:04 *AEDT* 2017 by *u5482878*
* Last modified *Mon Nov* 06 01:23:47 *AEDT* 2017 by *shamouda*
* Created *Wed Sep* 13 12:14:43 *AEST* 2017 by *u5482878*