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 and variables

CONSTANTS

PLACE,	The set of places
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
PROG_HOME,	
NTHREADS,	Minimum number of threads, more threads can be added up to *MXTHREADS* when running threads blocks
MXTHREADS,	
NBLOCKS,	
MXSTMTS,	
MUST_NOT_RUN	Validation constant: blocks that must not run, for example were not executed because of an execption

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

$Vars \triangleq \langle fstates,\ msgs,\ pstate,\ program,\ aseq,\ fseq,\ mseq,$
$readyQ,\ thrds,\ ppProgram,\ ppcurStmt,\ incPar,\ decPar\rangle$

Predicate to hide the finish implementation

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

INSTANCE $Commons$

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

Invariants (formulas true in every reachable state.)

$TypeOK \triangleq$
$\land\ fstates \in [IDRange \to FinishState]$
$\land\ readyQ \in [PLACE \to Seq(Activity)]$
$\land\ thrds \in [PLACE \to [ThreadID \to Thread]]$
$\land\ msgs \subseteq Messages$
$\land\ pstate \in \{\text{“init”},\ \text{“running”},\ \text{“terminated”},\ \text{“exceptionThrown”}\}$
$\land\ program \in [BlockID \to Block]$
$\land\ PROG\_HOME \in PLACE$
$\land\ aseq \in Nat$
$\land\ mseq \in Nat$
$\land\ fseq \in IDRange$
$\land\ ppcurStmt \in Nat$
$\land\ incPar \in [PLACE \to Nat]$
$\land\ decPar \in [PLACE \to Nat]$
$\land\ MUST\_NOT\_RUN \subseteq BlockID$

$PartialCorrectness \triangleq$
$\lor\ \land pstate = \text{“init”}$
$\land \forall p \in PLACE :$
$\land readyQ[p] = \langle\rangle$
$\land \forall t \in ThreadID : thrds[p][t].stack = \langle\rangle$
$\land \forall fid \in IDRange : fstates[fid].status = \text{“unused”}$
$\lor\ \land pstate \in \{\text{“terminated”},\ \text{“exceptionThrown”}\}$

$\land\ ppProgram = \langle\rangle$
$\land\ msgs = \{\}$
$\land\ \forall\, p \quad \in PLACE :$
$\quad \land\ readyQ[p] = \langle\rangle$
$\quad \land\ \forall\, t \in ThreadID : thrds[p][t].stack = \langle\rangle$
$\land\ \forall\, fid \in IDRange :$
$\quad \land\ fstates[fid].status \in \{\text{“unused”},\ \text{“forgotten”}\}$
$\land$ IF $pstate =$ “terminated”
$\quad$ THEN $\land\ fstates[FIRST\_ID].excs = \langle\rangle$
$\qquad \land\ \forall\, b \in BlockID : program[b].ran = 1$
$\quad$ ELSE $\land\ fstates[FIRST\_ID].excs \neq \langle\rangle$
$\qquad \land\ \forall\, b \in BlockID :$ IF $b \in MUST\_NOT\_RUN$
$\qquad\qquad$ THEN $program[b].ran = 0$
$\qquad\qquad$ ELSE $program[b].ran = 1$
$\lor\ \land\ pstate =$ “running”
$\quad \land\ ppProgram = \langle\rangle$
$\quad \land\ \lor\ \exists\, p \in PLACE :$
$\qquad \lor\ readyQ[p] \neq \langle\rangle$
$\qquad \lor\ \exists\, t \in ThreadID :\ \ thrds[p][t].stack \neq \langle\rangle$
$\quad \lor\ fstates[FIRST\_ID].status \neq$ “forgotten”

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

Initialization

$Init \triangleq$
$\land\ fstates = [r \in IDRange \mapsto$
$\quad [id \mapsto NotID,\ status \mapsto$ “unused”, $type \mapsto NotType,$
$\quad\ count \mapsto 0,\ excs \mapsto \langle\rangle,\ here \mapsto NotPlace,$
$\quad\ root \mapsto NotID,\ remActs \mapsto [p \in PLACE \mapsto 0]]]$
$\land\ readyQ = [p \in PLACE \mapsto \langle\rangle]$
$\land\ msgs = \{\}$
$\land\ pstate =$ “init”
$\land\ program = [b \in BlockID \mapsto$
$\quad [b \mapsto NotBlockID,\ type \mapsto$ “NA”, $dst \mapsto NotPlace,$
$\quad\ mxstmt \mapsto 0,\ stmts \mapsto [s \in StmtID \mapsto NotBlockID],$
$\quad\ ran \mapsto 0]]$
$\land\ aseq = 1$
$\land\ fseq = FIRST\_ID$
$\land\ mseq = 0$
$\land\ ppProgram = PROG$
$\land\ ppcurStmt = 0$
$\land\ incPar = [p \in PLACE \mapsto 0]$
$\land\ decPar = [p \in PLACE \mapsto 0]$

$$
\begin{array}{l}
\land\ thrds \quad = [p \in PLACE \mapsto \\
\qquad [t \in ThreadID \mapsto \\
\qquad\ \text{IF } t < NTHREADS \\
\qquad\quad \text{THEN } [tid \mapsto t,\ status \mapsto \text{“idle”},\ stack \mapsto \langle\rangle] \\
\qquad\quad \text{ELSE } [tid \mapsto t,\ status \mapsto \text{“NA”},\ stack \mapsto \langle\rangle]]]
\end{array}
$$

Parsing the input program into another format for easier processing

$$
\begin{array}{l}
ParseInputProgram \triangleq \\
\quad \land\ pstate = \text{“init”} \\
\quad \land\ Len(ppProgram) > 0 \\
\quad \land \text{LET } curBlk \triangleq Head(ppProgram) \\
\qquad\quad body \triangleq curBlk.body \\
\qquad\quad t \triangleq curBlk.type \\
\qquad\quad d \triangleq curBlk.dst \\
\qquad\quad b \triangleq curBlk.b \\
\qquad\quad h \triangleq \text{IF } body = \langle\rangle \text{ THEN } EMPTY\_BLOCK \text{ ELSE } Head(body) \\
\qquad \text{IN} \quad \land\ program' = [program \text{ EXCEPT } ![b].b = b, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad ![b].type\ \ = t, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad ![b].dst = d, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad ![b].mxstmt = ppcurStmt, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad ![b].ran = 0, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad ![b].stmts[ppcurStmt] = h] \\
\qquad\qquad\quad \land \text{IF } ((Len(body) = 0 \land ppcurStmt = 0) \lor Len(body) = 1) \\
\qquad\qquad\qquad \text{THEN } \land\ ppcurStmt' = 0 \\
\qquad\qquad\qquad\qquad\quad \land\ ppProgram' = Tail(ppProgram) \\
\qquad\qquad\qquad \text{ELSE } \land\ ppcurStmt' = ppcurStmt + 1 \\
\qquad\qquad\qquad\qquad\quad \land\ ppProgram' = \langle[type \mapsto t, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad dst \mapsto d, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad b \mapsto b, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad body \mapsto Tail(body), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad err \mapsto \text{“”}] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \rangle \circ Tail(ppProgram) \\
\quad \land \text{UNCHANGED } \langle fstates,\ pstate,\ msgs,\ aseq,\ fseq,\ mseq,\ readyQ, \\
\qquad\qquad\qquad thrds,\ incPar,\ decPar\rangle
\end{array}
$$

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

$$
\begin{array}{l}
Run \triangleq \\
\quad \land\ pstate = \text{“init”} \\
\quad \land\ Len(ppProgram) = 0 \\
\quad \land\ pstate' = \text{“running”} \\
\quad \land \text{LET } curStmt \triangleq \text{IF } program[0].type = \text{“finish”} \text{ THEN } -2 \text{ ELSE } -1 \\
\qquad \text{IN} \quad thrds' = [thrds \text{ EXCEPT } ![PROG\_HOME][0].stack = \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \langle[\ b\ \mapsto 0, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad i\ \mapsto curStmt, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad fid \mapsto NoParent]
\end{array}
$$

$\rangle$,
! $[PROG\_HOME][0].status =$ "running"]
$\wedge\ program' = [program$ EXCEPT $![0].ran = 1]$
$\wedge$ UNCHANGED $\langle fstates,\ msgs,\ aseq,\ fseq,\ mseq,\ readyQ,\ ppProgram,$
$ppcurStmt,\ incPar,\ decPar\rangle$

Scheduler Actions

Helper action: push activity

$PushReadyFIFO(here,\ activity) \triangleq$
$\wedge\ readyQ' = [readyQ$ EXCEPT $![here] = Append(@,\ activity)]$

Helper action: poll activity

$PollReadyFIFO(here) \triangleq$
$\wedge\ readyQ[here] \neq \langle\rangle$
$\wedge\ readyQ' = [readyQ$ EXCEPT $![here] = Tail(readyQ[here])]$

Push an activity received from another place

$SubmitRemoteActivity(here,\ src,\ activity) \triangleq$
$\wedge$ IF $activity.fid \neq NoParent$
THEN $Finish(activity.fid)!NotifyActivityCreation(src,\ activity)$
ELSE $fstates' = fstates$
$\wedge\ PushReadyFIFO(here,\ activity)$

Push a local activity

$SubmitLocalActivity(here,\ src,\ activity) \triangleq$
$\wedge$ IF $activity.fid \neq NoParent$
THEN $Finish(activity.fid)!NotifyActivitySpawnAndCreation(here,$
$src,$
$activity)$
ELSE $fstates' = fstates$
$\wedge\ PushReadyFIFO(here,\ activity)$

Increase the number of worker threads

$IncreaseParallelism(here) \triangleq$
$\wedge\ pstate =$ "running"
$\wedge\ incPar[here] > 0$
$\wedge$ LET $tid \triangleq FindThread(here,$ "NA")
IN $\wedge\ tid \neq NotThreadID$
$\wedge\ incPar' = [incPar$ EXCEPT $![here] = @ - 1]$
$\wedge\ thrds' = [thrds$ EXCEPT $![here][tid].status =$ "idle"]
$\wedge$ UNCHANGED $\langle fstates,\ msgs,\ pstate,\ program,\ aseq,\ fseq,\ mseq,\ readyQ,$
$ppProgram,\ ppcurStmt,\ decPar\rangle$

Decrease the number of worker threads

$DecreaseParallelism(here) \triangleq$

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

Program Execution Actions

Idle thread fetching a ready activity

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

Running thread processing an expression

$RThreadRunExpr(here, tid) \triangleq$
$\land\ pstate = \text{“running”}$
$\land\ thrds[here][tid].status = \text{“running”}$
$\land$ 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 $\langle fstates, msgs, pstate, program, aseq, fseq, mseq, readyQ, ppProgram, ppcurStmt, incPar, decPar\rangle$

Running thread processing the end of an *async* block

$RThreadRunAsyncEnd(here, tid) \triangleq$
$\land\ pstate = \text{“running”}$
$\land\ thrds[here][tid].status = \text{“running”}$
$\land$ 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 $![here][tid].stack = \langle\rangle,$
$![here][tid].status =$ "idle"$]$
ELSE $thrds' = [thrds$ EXCEPT $![here][tid].stack = Tail(@)]$
$\land$ IF $blk = 0$
THEN $pstate' =$ "terminated"
ELSE $pstate' = pstate$
$\land$ UNCHANGED $\langle msgs,\ pstate,\ program,\ aseq,\ fseq,\ mseq,\ readyQ,\ ppProgram,$
$ppcurStmt,\ incPar,\ decPar\rangle$

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

$RThreadRunFinishEnd(here,\ tid) \triangleq$
$\land$ $pstate =$ "running"
$\land$ $thrds[here][tid].status =$ "running"
$\land$ LET $top \triangleq Head(thrds[here][tid].stack)$
IN $\land$ $program[top.b].type =$ "finish"
$\land$ $program[top.b].mxstmt = top.i$
$\land$ $Finish(top.fid)!NotifyActivityTermination$
$\land$ $thrds' = [thrds$ EXCEPT $![here][tid].status =$ "blocked"$]$
$\land$ $incPar' = [incPar$ EXCEPT $![here] = @ + 1]$
$\land$ UNCHANGED $\langle msgs,\ pstate,\ program,\ aseq,\ fseq,\ mseq,\ readyQ,\ ppProgram,$
$ppcurStmt,\ decPar\rangle$

Terminated finish unblocks its thread

$BThreadUnblock(here,\ tid) \triangleq$
$\land$ $pstate =$ "running"
$\land$ $thrds[here][tid].status =$ "blocked"
$\land$ LET $top \triangleq Head(thrds[here][tid].stack)$
$blk \triangleq top.b$
$fid \triangleq top.fid$
IN $\land$ $program[blk].type =$ "finish"
$\land$ $program[blk].mxstmt = top.i$
$\land$ $Finish(fid)!Terminated$
$\land$ $decPar' = [decPar$ EXCEPT $![here] = @ + 1]$
$\land$ IF $Len(thrds[here][tid].stack) = 1$
THEN $thrds' = [thrds$ EXCEPT $![here][tid].stack = \langle\rangle,$
$![here][tid].status =$ "idle"$]$
ELSE $thrds' = [thrds$ EXCEPT $![here][tid].stack = Tail(@),$
$![here][tid].status =$ "running"$]$
$\land$ IF $blk = 0 \land Finish(fid)!HasExceptions$
THEN $pstate' =$ "exceptionThrown"
ELSE IF $blk = 0 \land \neg Finish(fid)!HasExceptions$
THEN $pstate' =$ "terminated"

$$
\begin{array}{ll}
& \text{ELSE} \;\; pstate' = pstate \\
\wedge & \text{UNCHANGED} \; \langle fstates,\, msgs,\, program,\, aseq,\, fseq,\, mseq,\, readyQ, \\
& \qquad ppProgram,\, ppcurStmt,\, incPar \rangle
\end{array}
$$

Running thread processing the beginning of a finish block

$$
\begin{array}{l}
RThreadRunFinishFirstStmt(here,\, tid) \;\triangleq \\
\quad \wedge\, pstate = \text{“running”} \\
\quad \wedge\, thrds[here][tid].status = \text{“running”} \\
\quad \wedge\, \text{LET} \;\; top \triangleq Head(thrds[here][tid].stack) \\
\qquad\qquad tail \triangleq Tail(thrds[here][tid].stack) \\
\qquad\qquad lstStmt \triangleq top.i \\
\qquad\qquad curStmt \triangleq top.i + 1 \\
\qquad\qquad blk \triangleq top.b \\
\qquad\qquad fid \triangleq top.fid \\
\qquad \text{IN} \;\; \wedge\, program[blk].type = \text{“finish”} \\
\qquad\qquad \wedge\, lstStmt = -2 \\
\qquad\qquad \wedge\, Finish(fseq)!Alloc(ROOT\_FINISH,\, here,\, fid) \\
\qquad\qquad \wedge\, thrds' = [thrds \;\text{EXCEPT}\; ![here][tid].stack = \langle [b \mapsto top.b, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad i \mapsto curStmt, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad fid \mapsto fseq] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \rangle \circ tail] \\
\qquad\qquad \wedge\, fseq' = fseq + 1 \\
\quad \wedge\, \text{UNCHANGED} \; \langle msgs,\, pstate,\, program,\, aseq,\, mseq,\, readyQ,\, ppProgram, \\
\qquad\qquad ppcurStmt,\, incPar,\, decPar \rangle
\end{array}
$$

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

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

$\land\ thrds' = [thrds \text{ EXCEPT } ![here][tid].stack =$
$\langle[\ b\ \mapsto top.b,$
$\ \ i\ \mapsto curStmt,$
$\ \ fid \mapsto fid]$
$\rangle \circ tail]$
$\land$ UNCHANGED $\langle msgs, pstate, program, fseq, mseq, ppProgram,$
$ppcurStmt, incPar, decPar\rangle$

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

$RThreadRunNestedRemoteAsync(here, tid) \triangleq$
$\land\ pstate = \text{“running”}$
$\land\ thrds[here][tid].status = \text{“running”}$
$\land$ LET $top \triangleq Head(thrds[here][tid].stack)$
$tail \triangleq Tail(thrds[here][tid].stack)$
$lstStmt \triangleq top.i$
$curStmt \triangleq top.i + 1$
$blk \triangleq top.b$
$fid \triangleq top.fid$
$root \triangleq GetRootFinishId(fid)$
$nested \triangleq program[blk].stmts[curStmt]$
IN $\land\ program[blk].type \neq \text{“expr”}$
$\land\ fid \neq NoParent$
$\land\ curStmt \geq 0$
$\land\ curStmt \leq program[blk].mxstmt$
$\land\ program[nested].type = \text{“async”}$
$\land\ program[nested].dst \neq here$
$\land\ \lor\ \land Finish(fid)!NotifySubActivitySpawn(program[nested].dst)$
$\land SendMsg([mid \mapsto mseq,$
$src\ \mapsto here,$
$dst\ \mapsto program[nested].dst,$
$type \mapsto \text{“async”},$
$fid \mapsto root,$
$b \mapsto nested])$
$\land\ mseq' = mseq + 1$
$\land\ thrds' = [thrds \text{ EXCEPT } ![here][tid].stack =$
$\langle[\ b\ \mapsto top.b,$
$\ \ i\ \mapsto curStmt,$
$\ \ fid \mapsto fid]$
$\rangle \circ tail]$
$\lor\ \land Finish(fid)!NotifySubActivitySpawnError(program[nested].dst)$
$\land\ msgs' = msgs$
$\land\ mseq' = mseq$
$\land\ thrds' = [thrds \text{ EXCEPT } ![here][tid].stack =$
$\langle[b\ \mapsto top.b,$
$\ \ i\ \mapsto program[blk].mxstmt,$

$fid \mapsto fid]$
$\rangle \circ tail]$
$\land$ UNCHANGED $\langle pstate,\ program,\ aseq,\ fseq,\ readyQ,\ ppProgram,$
$ppcurStmt,\ incPar,\ decPar\rangle$

Processing a nested finish in the currently running block

$RThreadRunNestedFinish(here,\ tid) \triangleq$
$\land\ pstate = \text{“running”}$
$\land\ thrds[here][tid].status = \text{“running”}$
$\land$ 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 $\land\ program[blk].type \neq \text{“expr”}$
$\land\ curStmt \geq 0$
$\land\ curStmt \leq program[blk].mxstmt$
$\land\ program[nested].type = \text{“finish”}$
$\land\ program[nested].dst = here$
$\land\ thrds' = [thrds$ EXCEPT $![here][tid].stack =$
$\langle[\ b \mapsto nested,$
$i \mapsto -2,$
$fid \mapsto fid],$
$[\ b \mapsto top.b,$
$i \mapsto curStmt,$
$fid \mapsto fid]$
$\rangle \circ tail]$
$\land\ program' = [program$ EXCEPT $![nested].ran = 1]$
$\land$ UNCHANGED $\langle fstates,\ msgs,\ pstate,\ aseq,\ fseq,\ mseq,\ readyQ,$
$ppProgram,\ ppcurStmt,\ incPar,\ decPar\rangle$

Processing a nested expression in the currently running block

$RThreadRunNestedExpr(here,\ tid) \triangleq$
$\land\ pstate = \text{“running”}$
$\land\ thrds[here][tid].status = \text{“running”}$
$\land$ 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 $\land\ program[blk].type \neq \text{“expr”}$

$\land\ curStmt \geq 0$
$\land\ curStmt \leq program[blk].mxstmt$
$\land\ program[nested].type = \text{“expr”}$
$\land\ program[nested].dst = here$
$\land\ thrds' = [thrds \text{ EXCEPT } ![here][tid].stack =$
$\langle[\ b \mapsto nested,$
$\ \ i \mapsto -1,$
$\ \ fid \mapsto fid],$
$\ [\ b \mapsto top.b,$
$\ \ i \mapsto curStmt,$
$\ \ fid \mapsto fid]$
$\rangle \circ tail]$
$\land\ program' = [program \text{ EXCEPT } ![nested].ran = 1]$
$\land$ UNCHANGED $\langle fstates, msgs, pstate, aseq, fseq, mseq, readyQ,$
$ppProgram, ppcurStmt, incPar, decPar\rangle$

Processing a nested error in the currently running block

$RThreadRunNestedError(here, tid) \triangleq$
$\land\ pstate = \text{“running”}$
$\land\ thrds[here][tid].status = \text{“running”}$
$\land$ 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 $\land\ program[blk].type \notin \{\text{“expr”}, \text{“error”}\}$
$\land\ curStmt \geq 0$
$\land\ curStmt \leq program[blk].mxstmt$
$\land\ program[nested].type = \text{“error”}$
$\land\ program[nested].dst = here$
$\land\ thrds' = [thrds \text{ EXCEPT } ![here][tid].stack =$ jump to the end of the current block
$\langle[\ b \mapsto top.b,$
$\ \ i \mapsto program[blk].mxstmt,$
$\ \ fid \mapsto fid]$
$\rangle \circ tail]$
$\land\ program' = [program \text{ EXCEPT } ![nested].ran = 1]$
$\land\ Finish(fid)!PushException([err \mapsto \text{“ErrorStmt”}, from \mapsto here])$
$\land$ UNCHANGED $\langle msgs, pstate, aseq, fseq, mseq, readyQ,$
$ppProgram, ppcurStmt, incPar, decPar\rangle$

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

$CreateRemoteFinish(here) \triangleq$
$\land\ pstate = \text{“running”}$

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

Pushing an incoming *async* to the ready queue

$RecvAsync(here) \triangleq$
$\land\ pstate = \text{“running”}$
$\land$ 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 $\land\ pid \neq NotID$
$\land\ fid \neq NotID$
$\land\ src \neq NotPlace$
$\land\ RecvMsg([mid \mapsto msg.mid,$
$src \mapsto msg.src,$
$dst \mapsto here,$
$type \mapsto \text{“async”},$
$b \mapsto blk,$
$fid \mapsto pid])$
$\land\ SubmitRemoteActivity(here, src, [aid \mapsto aseq,$
$b \mapsto blk,$
$fid \mapsto fid])$
$\land\ aseq' = aseq + 1$
$\land$ UNCHANGED $\langle pstate, program, fseq, mseq, thrds, ppProgram, ppcurStmt, incPar, decPar\rangle$

Enclosing finish receiving a termination signal from a remote task

$RecvAsyncTerm(here) \triangleq$
$\land\ pstate = \text{“running”}$
$\land$ 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 $\langle pstate, program, aseq, fseq, mseq, readyQ, thrds, ppProgram, ppcurStmt, incPar, decPar\rangle$

RemoteFinish notifying its *RootFinish* that it terminated

$$
\begin{array}{l}
NotifyParentFinish(fid) \triangleq \\
\quad \land\ pstate = \text{“running”} \\
\quad \land\ fstates[fid].status = \text{“finished”} \\
\quad \land\ \text{LET}\ type \triangleq fstates[fid].type \\
\qquad\qquad pid \triangleq fstates[fid].root \\
\qquad\qquad pidHome \triangleq GetFinishHome(pid) \\
\qquad\qquad here \triangleq fstates[fid].here \\
\qquad \text{IN}\ \ \text{IF}\ type = ROOT\_FINISH \\
\qquad\qquad\ \text{THEN}\ \land\ msgs' = msgs \\
\qquad\qquad\qquad\qquad \land\ mseq' = mseq \\
\qquad\qquad\qquad\qquad \land\ fstates' = [fstates\ \text{EXCEPT}\ ![fid].status = \text{“forgotten”}] \\
\qquad\qquad\ \text{ELSE}\ \ \land\ pidHome \neq here \\
\qquad\qquad\qquad\qquad \land\ Finish(fid)!SendTermMsg(mseq) \\
\qquad\qquad\qquad\qquad \land\ mseq' = mseq + 1 \\
\quad \land\ \text{UNCHANGED}\ \langle program,\ pstate,\ aseq,\ fseq,\ readyQ,\ thrds,\ ppProgram, \\
\qquad\qquad\qquad\qquad ppcurStmt,\ incPar,\ decPar\rangle
\end{array}
$$

Predicate enumerating all possible next actions

$$
\begin{array}{l}
Next \triangleq \\
\quad \lor\ ParseInputProgram \\
\quad \lor\ Run \\
\quad \lor\ \exists\, here \in PLACE : \\
\qquad \lor\ IncreaseParallelism(here) \\
\qquad \lor\ DecreaseParallelism(here) \\
\qquad \lor\ CreateRemoteFinish(here) \\
\qquad \lor\ RecvAsync(here) \\
\qquad \lor\ RecvAsyncTerm(here) \\
\qquad \lor\ \exists\, tid \in ThreadID : \\
\qquad\qquad \lor\ IThreadFetchActivity(here,\ tid) \\
\qquad\qquad \lor\ RThreadRunExpr(here,\ tid) \\
\qquad\qquad \lor\ RThreadRunAsyncEnd(here,\ tid) \\
\qquad\qquad \lor\ RThreadRunFinishEnd(here,\ tid) \\
\qquad\qquad \lor\ BThreadUnblock(here,\ tid) \\
\qquad\qquad \lor\ RThreadRunFinishFirstStmt(here,\ tid) \\
\qquad\qquad \lor\ RThreadRunNestedLocalAsync(here,\ tid) \\
\qquad\qquad \lor\ RThreadRunNestedRemoteAsync(here,\ tid) \\
\qquad\qquad \lor\ RThreadRunNestedFinish(here,\ tid) \\
\qquad\qquad \lor\ RThreadRunNestedExpr(here,\ tid) \\
\qquad\qquad \lor\ RThreadRunNestedError(here,\ tid) \\
\qquad \lor\ \exists\, fid \in IDRange : \\
\qquad\qquad \lor\ NotifyParentFinish(fid)
\end{array}
$$

Asserting fairness properties to all actions

$$
\begin{aligned}
&Liveness \triangleq \\
&\quad \land \mathrm{WF}_{Vars}(ParseInputProgram) \\
&\quad \land \mathrm{WF}_{Vars}(Run) \\
&\quad \land \forall\, here \in PLACE : \\
&\qquad \mathrm{WF}_{Vars}(IncreaseParallelism(here)) \\
&\qquad \land \mathrm{WF}_{Vars}(DecreaseParallelism(here)) \\
&\qquad \land \mathrm{WF}_{Vars}(CreateRemoteFinish(here)) \\
&\qquad \land \mathrm{WF}_{Vars}(RecvAsync(here)) \\
&\qquad \land \mathrm{WF}_{Vars}(RecvAsyncTerm(here)) \\
&\qquad \land \forall\, tid \in ThreadID : \\
&\qquad\quad \land \mathrm{WF}_{Vars}(IThreadFetchActivity(here,\ tid)) \\
&\qquad\quad \land \mathrm{WF}_{Vars}(RThreadRunExpr(here,\ tid)) \\
&\qquad\quad \land \mathrm{WF}_{Vars}(RThreadRunAsyncEnd(here,\ tid)) \\
&\qquad\quad \land \mathrm{WF}_{Vars}(RThreadRunFinishEnd(here,\ tid)) \\
&\qquad\quad \land \mathrm{WF}_{Vars}(BThreadUnblock(here,\ tid)) \\
&\qquad\quad \land \mathrm{WF}_{Vars}(RThreadRunFinishFirstStmt(here,\ tid)) \\
&\qquad\quad \land \mathrm{WF}_{Vars}(RThreadRunNestedLocalAsync(here,\ tid)) \\
&\qquad\quad \land \mathrm{WF}_{Vars}(RThreadRunNestedRemoteAsync(here,\ tid)) \\
&\qquad\quad \land \mathrm{WF}_{Vars}(RThreadRunNestedFinish(here,\ tid)) \\
&\qquad\quad \land \mathrm{WF}_{Vars}(RThreadRunNestedExpr(here,\ tid)) \\
&\qquad\quad \land \mathrm{WF}_{Vars}(RThreadRunNestedError(here,\ tid)) \\
&\quad \land \forall\, fid \in IDRange : \\
&\qquad \mathrm{WF}_{Vars}(NotifyParentFinish(fid))
\end{aligned}
$$

Specification

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

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

* Modification History
* Last modified *Fri Oct* 13 11:47:33 *AEDT* 2017 by *u5482878*
* Last modified *Tue Sep* 26 22:57:46 *AEST* 2017 by *shamouda*
* Created *Wed Sep* 13 12:14:43 *AEST* 2017 by *u5482878*