MODULE *Optimistic*

This specification models the 'optimistic *finish*′ protocol used for detecting the termination of async-finish task graphs. We model the graph as connected nodes of tasks. Finish objects do not represent separate nodes in the task graph, but implicit objects attached to tasks.

The model simulates all possible n-level task graphs that can be created on a p-place system, where each node of the task graph has c children. The variables *LEVEL*, *NUM_PLACES* and *WIDTH* can be used to configure the graph. The model also permits simulating 0 or more failures by configuring the *MAX_KILL* variable.

For the model checker to generate all possible execution scenarios, it can run out of memory, specially when activating failure recovery actions. We introduced the variables *KILL_FROM* and *KILL_TO* to control the range of steps at which failures can occur, so that we can cut the verification process into multiple phases. For example, we used 4 phases to simulate all possible execution scenarios for a 3-level 3-place task tree with width 2, that takes around 50 steps in total:
Phase 1: kills a place between steps 0 and 20.
Phase 2: kills a place between steps 20 and 30.
Phase 3: kills a place between steps 30 and 50.
Phase 4: kills a place between steps 50 and 100.

See the run figures at: https://*github.com*/shamouda/x10-formal-spec/tree/master/async-finish-optimistic/*run_figures*

EXTENDS *Integers*

CONSTANTS

LEVEL,	task tree levels
WIDTH,	task tree branching factor
NUM_PLACES,	number of places
MAX_KILL,	maximum number of failures to simulate
KILL_FROM, *KILL_TO*	the range of steps to simulate a failure at

VARIABLES

exec_state,	execution state
tasks,	set of tasks
f_set,	finish objects
lf_set,	local finish objects
rf_set,	resilient finish objects
msgs,	*msgs*
nxt_finish_id,	sequence of finish ids
nxt_task_id,	sequence of task ids
nxt_remote_place,	next place to communicate with
killed,	set of killed places
killed_cnt,	size of the killed set
rec_child,	pending recovery actions: ghosts queries
rec_to,	pending recovey actions: ignoring tasks to dead places
rec_from,	pending recovey actions: counting messages from dead places
rec_from_waiting,	pernding recovery actions: receiving counts of messages from dead places
lost_tasks,	debug variable: set of lost tasks due to a failure
lost_f_set,	debug variable: set of lost finishes
lost_lf_set,	debug variable: set of lost local finishes
step	the exectution step of the model

$Vars \triangleq \langle exec\_state,\ tasks,\ f\_set,\ lf\_set,\ rf\_set,\ msgs,$
$nxt\_finish\_id,\ nxt\_task\_id,\ nxt\_remote\_place,$
$killed,\ killed\_cnt,$
$lost\_tasks,\ lost\_f\_set,\ lost\_lf\_set,$
$rec\_child,\ rec\_to,\ rec\_from,\ rec\_from\_waiting,\ step\rangle$

$C \triangleq$ INSTANCE $OptimisticCommons$

$TypeOK \triangleq$

Variables type constrains

$\land\ exec\_state \in \{\text{“running”},\ \text{“success”}\}$
$\land\ tasks \subseteq C!Task$
$\land\ f\_set \subseteq C!Finish$
$\land\ lf\_set \subseteq C!LFinish$
$\land\ rf\_set \subseteq C!RFinish$
$\land\ nxt\_finish\_id \in C!FinishID$
$\land\ nxt\_task\_id \in C!TaskID$
$\land\ nxt\_remote\_place \in C!Place1D$
$\land\ killed \subseteq C!PlaceID$
$\land\ killed\_cnt \in 0\,..\,(NUM\_PLACES - 1)$
$\land\ rec\_child \subseteq C!GetChildrenTask$
$\land\ rec\_to \subseteq C!ConvTask$
$\land\ rec\_from \subseteq C!ConvTask$
$\land\ rec\_from\_waiting \subseteq C!ConvTask$
$\land\ step \in Nat$

$MustTerminate \triangleq$

Temporal property: the program must eventually terminate successfully

$\Diamond(exec\_state = \text{“success”})$

$Init \triangleq$

Initialize variables

$\land\ exec\_state = \text{“running”}$
$\land\ tasks = \{C!RootTask,\ C!RootFinishTask\}$
$\land\ f\_set = \{C!RootFinish\}$
$\land\ lf\_set = \{\}$
$\land\ rf\_set = \{\}$
$\land\ msgs = \{\}$
$\land\ nxt\_finish\_id = 2$
$\land\ nxt\_task\_id = 2$
$\land\ nxt\_remote\_place = [i \in C!PlaceID \mapsto i]$
$\land\ killed = \{\}$

$\land\ killed\_cnt = 0$
$\land\ lost\_tasks = \{\}$
$\land\ lost\_f\_set = \{\}$
$\land\ lost\_lf\_set = \{\}$
$\land\ rec\_child = \{\}$
$\land\ rec\_to = \{\}$
$\land\ rec\_from = \{\}$
$\land\ rec\_from\_waiting = \{\}$
$\land\ step = 0$

Utility actions: creating instances of task, finish, resilient finish and local finish

$NewFinish(task) \triangleq$
$[id \mapsto nxt\_finish\_id,$
$\ pred\_id \mapsto task.id,$
$\ home \mapsto task.dst,$
$\ origin \mapsto task.src,$
$\ parent\_finish\_id \mapsto task.finish\_id,$
$\ status \mapsto$ "active",
$\ lc \mapsto 1$ the main finish task
$]$

$NewResilientFinish(finish) \triangleq$
$[id \mapsto finish.id,$
$\ home \mapsto finish.home,$
$\ origin \mapsto finish.origin,$
$\ parent\_finish\_id \mapsto finish.parent\_finish\_id,$
$\ transOrLive \mapsto C!Place2DInitResilientFinish(finish.home),$
$\ sent \mapsto C!Place2DInitResilientFinish(finish.home),$
$\ gc \mapsto 1,$
$\ ghost\_children \mapsto \{\},$
$\ isAdopted \mapsto$ FALSE$]$

$NewLocalFinish(fid,\ dst) \triangleq$
$[id \mapsto fid,$
$\ home \mapsto dst,$
$\ lc \mapsto 0,$
$\ reported \mapsto C!Place1DZeros,$
$\ received \mapsto C!Place1DZeros,$
$\ deny \mapsto C!Place1DZeros]$

$NewTask(pred,\ fid,\ s,\ d,\ t,\ l,\ st,\ fin\_type) \triangleq$
$[id \mapsto nxt\_task\_id,$
$\ pred\_id \mapsto pred,$
$\ src \mapsto s,$
$\ dst \mapsto d,$

$finish\_id \mapsto fid$,
$level \mapsto l$,
$last\_branch \mapsto 0$,
$status \mapsto st$,
$type \mapsto t$,
$finish\_type \mapsto fin\_type]$

Finish Actions

$Task\_CreatingFinish \triangleq$
$\land\ exec\_state = \text{“running”}$
$\land$ LET $task \triangleq C!FindRunningTask(LEVEL - 1)$
$task\_updated \triangleq$ IF $task = C!NOT\_TASK$ THEN $C!NOT\_TASK$
ELSE $[task$ EXCEPT $!.last\_branch = task.last\_branch + 1,$
$!.status = \text{“blocked”}]$
$finish \triangleq$ IF $task \neq C!NOT\_TASK$
THEN $NewFinish(task)$
ELSE $C!NOT\_FINISH$
$finish\_task \triangleq$ IF $task \neq C!NOT\_TASK$
THEN $NewTask(task.id,\ finish.id,\ task.dst,\ task.dst,$
$\text{“finishMainTask”},\ task.level + 1,\ \text{“running”},\ \text{“global”})$
ELSE $C!NOT\_TASK$
IN $\land\ task \neq C!NOT\_TASK$
$\land\ nxt\_finish\_id' = nxt\_finish\_id + 1$
$\land\ nxt\_task\_id' = nxt\_task\_id + 1$
$\land\ f\_set' = f\_set \cup \{finish\}$
$\land\ tasks' = (tasks \setminus \{task\}) \cup \{task\_updated,\ finish\_task\}$
$\land\ step' = step + 1$
$\land$ UNCHANGED $\langle exec\_state,\ lf\_set,\ rf\_set,\ msgs,$
$nxt\_remote\_place,$
$killed,\ killed\_cnt,$
$lost\_tasks,\ lost\_f\_set,\ lost\_lf\_set,$
$rec\_child,\ rec\_to,\ rec\_from,\ rec\_from\_waiting\rangle$

$Finish\_CreatingRemoteTask \triangleq$
$\land\ exec\_state = \text{“running”}$
$\land$ LET $task \triangleq C!FindRunningTaskWithFinishType(LEVEL - 1,\ \text{“global”})$
$task\_updated \triangleq$ IF $task = C!NOT\_TASK$ THEN $C!NOT\_TASK$
ELSE $[task$ EXCEPT $!.last\_branch = task.last\_branch + 1,$
$!.status = \text{“blocked”}]$
$finish \triangleq$ IF $task = C!NOT\_TASK$ THEN $C!NOT\_FINISH$
ELSE $C!FindFinishById(task.finish\_id)$
$new\_finish\_status \triangleq$ IF $C!IsPublished(task.finish\_id)$
THEN $finish.status$
ELSE $\text{“waitingForPublish”}$

$finish\_updated \triangleq$ IF $task = C!NOT\_TASK$ THEN $C!NOT\_FINISH$
ELSE $[finish$ EXCEPT $!.status = new\_finish\_status]$
$src \triangleq task.dst$
$dst \triangleq C!NextRemotePlace(src)$
$new\_task\_status \triangleq$ IF $C!IsPublished(task.finish\_id)$
THEN "waitingForTransit"
ELSE "waitingForPublish"
$new\_task \triangleq$ IF $task = C!NOT\_TASK$ THEN $C!NOT\_TASK$
ELSE $NewTask(task.id,\ task.finish\_id,\ src,\ dst,$ "normal", $task.level + 1,\ new\_task\_sta$
$msg\_transit \triangleq [from \mapsto$ "src", $to \mapsto$ "rf", $tag \mapsto$ "transit",
$src \mapsto new\_task.src,\ dst \mapsto new\_task.dst,$
$finish\_id \mapsto new\_task.finish\_id,$
$task\_id \mapsto new\_task.id]$
$msg\_publish \triangleq [from \mapsto$ "f", $to \mapsto$ "rf", $tag \mapsto$ "publish",
$src \mapsto finish.home,$
$finish\_id \mapsto finish.id]$
IN $\land\ task \neq C!NOT\_TASK$
$\land\ finish.status =$ "active"
$\land\ nxt\_task\_id' = nxt\_task\_id + 1$
$\land\ tasks' = (tasks \setminus \{task\}) \cup \{task\_updated,\ new\_task\}$
$\land\ f\_set' = (f\_set \setminus \{finish\}) \cup \{finish\_updated\}$
$\land\ C!ShiftNextRemotePlace(src)$
$\land$ IF $C!IsPublished(task.finish\_id)$
THEN $C!SendMsg(msg\_transit)$
ELSE $C!SendMsg(msg\_publish)$
$\land\ step' = step + 1$
$\land$ UNCHANGED $\langle exec\_state,\ lf\_set,\ rf\_set,$
$nxt\_finish\_id,$
$killed,\ killed\_cnt,$
$lost\_tasks,\ lost\_f\_set,\ lost\_lf\_set,$
$rec\_child,\ rec\_to,\ rec\_from,\ rec\_from\_waiting\rangle$

$Finish\_ReceivingPublishDoneSignal \triangleq$
$\land\ exec\_state =$ "running"
$\land$ LET $msg \triangleq C!FindMessageToActivePlaceWithTag($"f", "publishDone"$)$
$finish \triangleq$ IF $msg = C!NOT\_MESSAGE$ THEN $C!NOT\_FINISH$
ELSE $C!FindFinishById(msg.finish\_id)$
$finish\_updated \triangleq$ IF $msg = C!NOT\_MESSAGE$ THEN $C!NOT\_FINISH$
ELSE $[finish$ EXCEPT $!.status =$ "active"$]$
$pending\_task \triangleq C!FindPendingRemoteTask(finish.id,$ "waitingForPublish"$)$
$pending\_task\_updated \triangleq$ IF $msg = C!NOT\_MESSAGE$ THEN $C!NOT\_TASK$
ELSE $[pending\_task$ EXCEPT $!.status =$ "waitingForTransit"$]$
$msg\_transit \triangleq [from \mapsto$ "src", $to \mapsto$ "rf", $tag \mapsto$ "transit",
$src \mapsto pending\_task.src,\ dst \mapsto pending\_task.dst,$
$finish\_id \mapsto pending\_task.finish\_id,$

```
                                task_id ↦ pending_task.id]
    IN    ∧ msg ≠ C!NOT_MESSAGE
          ∧ C!ReplaceMsg(msg, msg_transit)
          ∧ f_set' = (f_set \ {finish}) ∪ {finish_updated}
          ∧ tasks' = (tasks \ {pending_task}) ∪ {pending_task_updated}
          ∧ step' = step + 1
  ∧ UNCHANGED ⟨exec_state, lf_set, rf_set,
                nxt_finish_id, nxt_task_id, nxt_remote_place,
                killed, killed_cnt,
                lost_tasks, lost_f_set, lost_lf_set,
                rec_child, rec_to, rec_from, rec_from_waiting⟩

Finish_TerminatingTask ≜
  ∧ exec_state = "running"
  ∧ LET task ≜ C!FindTaskToTerminate("global")
        finish ≜ IF task = C!NOT_TASK THEN C!NOT_FINISH
                 ELSE C!FindFinishById(task.finish_id)
        task_updated ≜ IF task ≠ C!NOT_TASK
                       THEN [task EXCEPT !.status = "terminated"]
                       ELSE C!NOT_TASK
        finish_updated ≜ IF task = C!NOT_TASK THEN C!NOT_FINISH
                         ELSE IF finish.lc = 1 ∧ C!IsPublished(finish.id)
                         THEN [finish EXCEPT !.lc = finish.lc − 1,
                                             !.status = "waitingForRelease"]
                         ELSE IF finish.lc = 1 ∧ ¬C!IsPublished(finish.id)
                         THEN [finish EXCEPT !.lc = finish.lc − 1,
                                             !.status = "released"]
                         ELSE [finish EXCEPT !.lc = finish.lc − 1]
    IN    ∧ task ≠ C!NOT_TASK
          ∧ finish ≠ C!NOT_FINISH
          ∧ f_set' = (f_set \ {finish}) ∪ {finish_updated}
          ∧ IF finish_updated.status = "waitingForRelease"
            THEN msgs' = msgs ∪ {[from ↦ "f", to ↦ "rf", tag ↦ "terminateTask",
                                  src ↦ finish.home,
                                  finish_id ↦ finish.id,
                                  task_id ↦ task.id,
                                  term_tasks_by_src ↦ C!Place1DTerminateTask(finish.home, 1),
                                  term_tasks_dst ↦ finish.home]}
            ELSE msgs' = msgs
          ∧ IF finish_updated.status = "released"
            THEN LET task_blocked ≜ C!FindBlockedTask(finish.pred_id)
                     task_unblocked ≜ [task_blocked EXCEPT !.status = "running"]
                 IN tasks' = (tasks \ {task, task_blocked}) ∪ {task_updated, task_unblocked}
            ELSE tasks' = (tasks \ {task}) ∪ {task_updated}
          ∧ step' = step + 1
```

$\land$ UNCHANGED $\langle exec\_state,\ lf\_set,\ rf\_set,$
$nxt\_finish\_id,\ nxt\_task\_id,\ nxt\_remote\_place,$
$killed,\ killed\_cnt,$
$lost\_tasks,\ lost\_f\_set,\ lost\_lf\_set,$
$rec\_child,\ rec\_to,\ rec\_from,\ rec\_from\_waiting\rangle$

$Finish\_ReceivingReleaseSignal \triangleq$
$\land\ exec\_state = \text{“running”}$
$\land$ LET $msg \triangleq C!FindMessageToActivePlaceWithTag(\text{“f”},\ \text{“release”})$
$finish \triangleq$ IF $msg = C!NOT\_MESSAGE$ THEN $C!NOT\_FINISH$
ELSE $C!FindFinishToRelease(msg.finish\_id)$
$finish\_updated \triangleq$ IF $msg = C!NOT\_MESSAGE$ THEN $C!NOT\_FINISH$
ELSE $[finish$ EXCEPT $!.status = \text{“released”}]$
$task\_blocked \triangleq$ IF $msg = C!NOT\_MESSAGE$ THEN $C!NOT\_TASK$
ELSE $C!FindBlockedTask(finish.pred\_id)$
$task\_unblocked \triangleq$ IF $msg = C!NOT\_MESSAGE$ THEN $C!NOT\_TASK$
ELSE $[task\_blocked$ EXCEPT $!.status = \text{“running”}]$
IN $\land\ msg \neq C!NOT\_MESSAGE$
$\land\ C!RecvMsg(msg)$
$\land\ f\_set' = (f\_set \setminus \{finish\}) \cup \{finish\_updated\}$
$\land\ tasks' = (tasks \setminus \{task\_blocked\}) \cup \{task\_unblocked\}$
$\land\ step' = step + 1$
$\land$ UNCHANGED $\langle exec\_state,\ lf\_set,\ rf\_set,$
$nxt\_finish\_id,\ nxt\_task\_id,\ nxt\_remote\_place,$
$killed,\ killed\_cnt,$
$lost\_tasks,\ lost\_f\_set,\ lost\_lf\_set,$
$rec\_child,\ rec\_to,\ rec\_from,\ rec\_from\_waiting\rangle$

Actions applicable to *Finish* and Local *Finish*

$DroppingTask \triangleq$
$\land\ exec\_state = \text{“running”}$
$\land$ LET $msg \triangleq C!FindMessageToActivePlaceWithTag(\text{“src”},\ \text{“transitNotDone”})$
$task \triangleq$ IF $msg = C!NOT\_MESSAGE$ THEN $C!NOT\_TASK$
ELSE $C!FindTaskById(msg.task\_id)$
$task\_updated \triangleq$ IF $task = C!NOT\_TASK$ THEN $C!NOT\_TASK$
ELSE $[task$ EXCEPT $!.status = \text{“dropped”}]$
$blocked\_task \triangleq C!FindTaskById(task.pred\_id)$
$blocked\_task\_updated \triangleq [blocked\_task$ EXCEPT $!.status = \text{“running”}]$
IN $\land\ msg \neq C!NOT\_MESSAGE$
$\land\ task.status = \text{“waitingForTransit”}$
$\land\ blocked\_task.status = \text{“blocked”}$
$\land\ tasks' = (tasks \setminus \{task,\ blocked\_task\}) \cup \{task\_updated,\ blocked\_task\_updated\}$
$\land\ C!RecvMsg(msg)$
$\land\ step' = step + 1$

$\land$ UNCHANGED $\langle exec\_state,\ f\_set,\ lf\_set,\ rf\_set,$
$nxt\_finish\_id,\ nxt\_task\_id,\ nxt\_remote\_place,$
$killed,\ killed\_cnt,$
$lost\_tasks,\ lost\_f\_set,\ lost\_lf\_set,$
$rec\_child,\ rec\_to,\ rec\_from,\ rec\_from\_waiting\rangle$

$SendingTask \triangleq$
$\land\ exec\_state =$ "running"
$\land$ LET $msg \triangleq C!FindMessageToActivePlaceWithTag($"src", "transitDone"$)$
$task \triangleq$ IF $msg = C!NOT\_MESSAGE$ THEN $C!NOT\_TASK$
ELSE $C!FindTaskById(msg.task\_id)$
$task\_updated \triangleq$ IF $task = C!NOT\_TASK$ THEN $C!NOT\_TASK$
ELSE $[task$ EXCEPT $!.status =$ "sent"$]$
$blocked\_task \triangleq C!FindTaskById(task.pred\_id)$
$blocked\_task\_updated \triangleq [blocked\_task$ EXCEPT $!.status =$ "running"$]$
IN $\land\ msg \neq C!NOT\_MESSAGE$
$\land\ task.status =$ "waitingForTransit"
$\land\ blocked\_task.status =$ "blocked"
$\land\ tasks' = (tasks \setminus \{task,\ blocked\_task\}) \cup \{task\_updated,\ blocked\_task\_updated\}$
$\land\ C!ReplaceMsg(msg, [from \mapsto$ "src", $to \mapsto$ "dst", $tag \mapsto$ "task",
$src \mapsto task.src,\ dst \mapsto task.dst,$
$finish\_id \mapsto task.finish\_id,$
$task\_id \mapsto task.id])$
$\land\ step' = step + 1$
$\land$ UNCHANGED $\langle exec\_state,\ f\_set,\ lf\_set,\ rf\_set,$
$nxt\_finish\_id,\ nxt\_task\_id,\ nxt\_remote\_place,$
$killed,\ killed\_cnt,$
$lost\_tasks,\ lost\_f\_set,\ lost\_lf\_set,$
$rec\_child,\ rec\_to,\ rec\_from,\ rec\_from\_waiting\rangle$

$ReceivingTask \triangleq$
$\land\ exec\_state =$ "running"
$\land$ LET $msg \triangleq C!FindMessageToActivePlaceWithTag($"dst", "task"$)$
$src \triangleq msg.src$
$dst \triangleq msg.dst$
$finish\_id \triangleq msg.finish\_id$
$lfinish \triangleq$ IF $msg = C!NOT\_MESSAGE$ THEN $C!NOT\_FINISH$
ELSE IF $C!LocalFinishExists(dst,\ finish\_id)$ THEN $C!FindLocalFinish(dst,\ finish\_id)$
ELSE $NewLocalFinish(finish\_id,\ dst)$
$lfinish\_updated \triangleq [lfinish$ EXCEPT $!.received[src] = lfinish.received[src] + 1,$
$!.lc = lfinish.lc + 1]$
$task \triangleq$ IF $msg = C!NOT\_MESSAGE$ THEN $C!NOT\_TASK$
ELSE $C!FindTaskById(msg.task\_id)$
$task\_updated \triangleq$ IF $task = C!NOT\_TASK$ THEN $C!NOT\_TASK$
ELSE $[task$ EXCEPT $!.status =$ "running"$]$

```
    IN   ∧ msg ≠ C!NOT_MESSAGE
         ∧ C!RecvMsg(msg)
         ∧ IF lfinish.deny[src] = 1
             THEN ∧ lf_set' = lf_set
                  ∧ tasks' = tasks
             ELSE ∧ lf_set' = (lf_set \ {lfinish}) ∪ {lfinish_updated}
                  ∧ tasks' = (tasks \ {task}) ∪ {task_updated}
         ∧ step' = step + 1
  ∧ UNCHANGED ⟨exec_state, f_set, rf_set,
               nxt_finish_id, nxt_task_id, nxt_remote_place,
               killed, killed_cnt,
               lost_tasks, lost_f_set, lost_lf_set,
               rec_child, rec_to, rec_from, rec_from_waiting⟩
```

Local *Finish* Actions

```
LocalFinish_CreatingRemoteTask ≜   create task with status created and put it in the set
  ∧ exec_state = "running"
  ∧ LET task ≜ C!FindRunningTaskWithFinishType(LEVEL − 1, "local")
        task_updated ≜ IF task = C!NOT_TASK THEN C!NOT_TASK
                       ELSE [task EXCEPT !.last_branch = task.last_branch + 1,
                                         !.status = "blocked"]
        finish ≜ IF task = C!NOT_TASK THEN C!NOT_FINISH
                 ELSE C!FindFinishById(task.finish_id)
        src ≜ task.dst
        dst ≜ C!NextRemotePlace(src)
        new_task ≜ IF task = C!NOT_TASK THEN C!NOT_TASK
                   ELSE NewTask(task.id, task.finish_id, src, dst, "normal", task.level + 1, "waitingForTr
        msg_transit ≜ [from ↦ "src", to ↦ "rf", tag ↦ "transit",
                       src ↦ new_task.src, dst ↦ new_task.dst,
                       finish_id ↦ new_task.finish_id,
                       task_id ↦ new_task.id]
    IN  ∧ task ≠ C!NOT_TASK
        ∧ nxt_task_id' = nxt_task_id + 1
        ∧ tasks' = (tasks \ {task}) ∪ {task_updated, new_task}
        ∧ C!ShiftNextRemotePlace(src)
        ∧ C!SendMsg(msg_transit)
        ∧ step' = step + 1
  ∧ UNCHANGED ⟨exec_state, f_set, lf_set, rf_set,
               nxt_finish_id,
               killed, killed_cnt,
               lost_tasks, lost_f_set, lost_lf_set,
               rec_child, rec_to, rec_from, rec_from_waiting⟩

LocalFinish_TerminatingTask ≜
```

```
  ∧ exec_state = "running"
  ∧ LET task ≜ C!FindTaskToTerminate("local")
        task_updated ≜ IF task = C!NOT_TASK THEN C!NOT_TASK
                       ELSE [task EXCEPT !.status = "terminated"]
        here ≜ task.dst
        finish_id ≜ task.finish_id
        lfinish ≜ IF task = C!NOT_TASK THEN C!NOT_FINISH
                  ELSE C!FindLocalFinish(here, finish_id)
        lfinish_updated ≜ IF task = C!NOT_TASK THEN C!NOT_FINISH
                          ELSE [lfinish EXCEPT !.lc = lfinish.lc − 1]
        term_tasks ≜ IF task = C!NOT_TASK THEN C!NOT_FINISH
                     ELSE [i ∈ C!PlaceID ↦ IF i = lfinish.home THEN 0
                                           ELSE lfinish.received[i] − lfinish.reported[i] ]
        lfinish_terminated ≜ IF task = C!NOT_TASK THEN C!NOT_FINISH
                             ELSE [lfinish EXCEPT !.lc = 0,
                                                  !.reported = lfinish.received]
    IN  ∧ task ≠ C!NOT_TASK
        ∧ lfinish ≠ C!NOT_FINISH
        ∧ IF lfinish_updated.lc = 0
          THEN ∧ msgs' = msgs ∪ {[from ↦ "f", to ↦ "rf", tag ↦ "terminateTask",
                                  src ↦ here,
                                  finish_id ↦ finish_id,
                                  task_id ↦ task.id,
                                  term_tasks_by_src ↦ term_tasks,
                                  term_tasks_dst ↦ here]}
               ∧ lf_set' = (lf_set \ {lfinish}) ∪ {lfinish_terminated}
          ELSE ∧ msgs' = msgs
               ∧ lf_set' = (lf_set \ {lfinish}) ∪ {lfinish_updated}
        ∧ tasks' = (tasks \ {task}) ∪ {task_updated}
        ∧ step' = step + 1
  ∧ UNCHANGED ⟨exec_state, f_set, rf_set,
               nxt_finish_id, nxt_task_id, nxt_remote_place,
               killed, killed_cnt,
               lost_tasks, lost_f_set, lost_lf_set,
               rec_child, rec_to, rec_from, rec_from_waiting⟩

LocalFinish_MarkingDeadPlace ≜
  ∧ exec_state = "running"
  ∧ LET msg ≜ C!FindMessageToActivePlaceWithTag("dst", "countDropped")
        finish_id ≜ msg.finish_id
        here ≜ msg.dst
        dead ≜ msg.src
        lfinish ≜ IF msg = C!NOT_MESSAGE THEN C!NOT_FINISH
                  ELSE IF C!LocalFinishExists(here, finish_id) THEN C!FindLocalFinish(here, finish_id)
                  ELSE NewLocalFinish(finish_id, here)
```

$lfinish\_updated \triangleq$ IF $msg = C!NOT\_MESSAGE$ THEN $C!NOT\_FINISH$
ELSE $[lfinish$ EXCEPT $!.deny[dead] = 1]$
$resp \triangleq$ IF $msg = C!NOT\_MESSAGE$ THEN $C!NOT\_MESSAGE$
ELSE $[from \mapsto$ "dst", $to \mapsto$ "rf", $tag \mapsto$ "countDroppedDone",
$finish\_id \mapsto msg.finish\_id$,
$src \mapsto msg.src$, $dst \mapsto msg.dst$,
$num\_dropped \mapsto msg.num\_sent - lfinish.received[dead]]$
IN $\land msg \neq C!NOT\_MESSAGE$
$\land C!ReplaceMsg(msg, resp)$
$\land lf\_set' = (lf\_set \setminus \{lfinish\}) \cup \{lfinish\_updated\}$
$\land step' = step + 1$
$\land$ UNCHANGED $\langle exec\_state, tasks, f\_set, rf\_set,$
$nxt\_finish\_id, nxt\_task\_id, nxt\_remote\_place,$
$killed, killed\_cnt,$
$lost\_tasks, lost\_f\_set, lost\_lf\_set,$
$rec\_child, rec\_to, rec\_from, rec\_from\_waiting\rangle$

Resilient Store Actions

$Store\_ReceivingPublishSignal \triangleq$
$\land exec\_state =$ "running"
$\land$ LET $msg \triangleq C!FindMessageToActivePlaceWithTag($"rf", "publish"$)$
$finish\_home \triangleq msg.src$
$finish \triangleq$ IF $msg = C!NOT\_MESSAGE \lor finish\_home \in killed$
THEN $C!NOT\_FINISH$
ELSE $C!FindFinishById(msg.finish\_id)$
IN $\land msg \neq C!NOT\_MESSAGE$
$\land$ IF $finish\_home \notin killed$
THEN $\land C!ReplaceMsg(msg, [from \mapsto$ "rf", $to \mapsto$ "f", $tag \mapsto$ "publishDone",
$dst \mapsto msg.src,$
$finish\_id \mapsto msg.finish\_id])$
$\land rf\_set' = rf\_set \cup \{NewResilientFinish(finish)\}$
ELSE $\land C!RecvMsg(msg)$
$\land rf\_set' = rf\_set$
$\land step' = step + 1$
$\land$ UNCHANGED $\langle exec\_state, tasks, f\_set, lf\_set,$
$nxt\_finish\_id, nxt\_task\_id, nxt\_remote\_place,$
$killed, killed\_cnt,$
$lost\_tasks, lost\_f\_set, lost\_lf\_set,$
$rec\_child, rec\_to, rec\_from, rec\_from\_waiting\rangle$

$Store\_ReceivingTransitSignal \triangleq$
$\land exec\_state =$ "running"
$\land$ LET $msg \triangleq C!FindMessageToActivePlaceWithTag($"rf", "transit"$)$
$rf \triangleq$ IF $msg = C!NOT\_MESSAGE$ THEN $C!NOT\_FINISH$

```
                ELSE  C!FindResilientFinishById(msg.finish_id)
        s ≜ msg.src
        d ≜ msg.dst
        rf_updated ≜ IF msg = C!NOT_MESSAGE THEN C!NOT_FINISH
                     ELSE [rf EXCEPT !.sent[s][d] = rf.sent[s][d] + 1,
                                     !.transOrLive[s][d] = rf.transOrLive[s][d] + 1,
                                     !.gc = rf.gc + 1]
        msg_tag ≜ IF s ∈ killed ∨ d ∈ killed THEN "transitNotDone" ELSE "transitDone"
    IN  ∧ msg ≠ C!NOT_MESSAGE
        ∧ ¬C!IsRecoveringTasksToDeadPlaces(rf.id)
        ∧ IF s ∈ killed ∨ d ∈ killed
            THEN rf_set' = rf_set
            ELSE rf_set' = (rf_set \ {rf}) ∪ {rf_updated}
        ∧ C!ReplaceMsg(msg, [from ↦ "rf", to ↦ "src", tag ↦ msg_tag,
                             dst ↦ s,
                             finish_id ↦ msg.finish_id,
                             task_id ↦ msg.task_id])
        ∧ step' = step + 1
  ∧ UNCHANGED ⟨exec_state, tasks, f_set, lf_set,
               nxt_finish_id, nxt_task_id, nxt_remote_place,
               killed, killed_cnt,
               lost_tasks, lost_f_set, lost_lf_set,
               rec_child, rec_to, rec_from, rec_from_waiting⟩

Store_ReceivingTerminateTaskSignal ≜
  ∧ exec_state = "running"
  ∧ LET msg ≜ C!FindMessageToActivePlaceWithTag("rf", "terminateTask")
        term_tasks ≜ msg.term_tasks_by_src
        dst ≜ msg.term_tasks_dst
        rf ≜ IF msg = C!NOT_MESSAGE ∨ dst ∈ killed THEN C!NOT_FINISH
             ELSE C!FindResilientFinishById(msg.finish_id)
        trans_live_updated ≜ [i ∈ C!PlaceID ↦ [j ∈ C!PlaceID ↦
                               IF j = dst THEN rf.transOrLive[i][j] - term_tasks[i]
                                ELSE rf.transOrLive[i][j]
                              ]]
        total ≜ C!Sum(term_tasks)
        rf_updated ≜ IF msg = C!NOT_MESSAGE ∨ dst ∈ killed THEN C!NOT_FINISH
                     ELSE [rf EXCEPT !.transOrLive     = trans_live_updated,
                                     !.gc = rf.gc - total]
    IN  ∧ msg ≠ C!NOT_MESSAGE
        ∧ total ≠ -1  see C!Sum() definition
        ∧ IF dst ∉ killed
            THEN ∧ ¬C!IsRecoveringTasksToDeadPlaces(rf.id)
                 ∧ C!ApplyTerminateSignal(rf, rf_updated, msg)
            ELSE C!RecvTerminateSignal(msg)
```

$\land\ step' = step + 1$
$\land$ UNCHANGED $\langle exec\_state,\ tasks,\ f\_set,\ lf\_set,$
$nxt\_finish\_id,\ nxt\_task\_id,\ nxt\_remote\_place,$
$killed,\ killed\_cnt,$
$lost\_tasks,\ lost\_f\_set,\ lost\_lf\_set,$
$rec\_child,\ rec\_to,\ rec\_from,\ rec\_from\_waiting\rangle$

$Store\_ReceivingTerminateGhostSignal \triangleq$
$\land\ exec\_state =$ "running"
$\land$ LET $msg \triangleq C!FindMessageToActivePlaceWithTag($"rf", "terminateGhost"$)$
$rf \triangleq$ IF $msg = C!NOT\_MESSAGE$ THEN $C!NOT\_FINISH$
ELSE $C!FindResilientFinishById(msg.finish\_id)$
$ghost\_child \triangleq msg.ghost\_finish\_id$
$rf\_updated \triangleq$ IF $msg = C!NOT\_MESSAGE$ THEN $C!NOT\_FINISH$
ELSE $[rf$ EXCEPT $!.ghost\_children = rf.ghost\_children \setminus \{ghost\_child\}]$
IN $\land\ msg \neq C!NOT\_MESSAGE$
$\land\ \neg C!IsRecoveringTasksToDeadPlaces(rf.id)$
$\land\ C!ApplyTerminateSignal(rf,\ rf\_updated,\ msg)$
$\land\ step' = step + 1$
$\land$ UNCHANGED $\langle exec\_state,\ tasks,\ f\_set,\ lf\_set,$
$nxt\_finish\_id,\ nxt\_task\_id,\ nxt\_remote\_place,$
$killed,\ killed\_cnt,$
$lost\_tasks,\ lost\_f\_set,\ lost\_lf\_set,$
$rec\_child,\ rec\_to,\ rec\_from,\ rec\_from\_waiting\rangle$

$Store\_FindingGhostChildren \triangleq$
$\land\ exec\_state =$ "running"
$\land$ LET $req \triangleq C!FindMarkGhostChildrenRequest$
$rf \triangleq$ IF $req = C!NOT\_REQUEST$ THEN $C!NOT\_FINISH$
ELSE $C!FindResilientFinishById(req.finish\_id)$
$ghosts \triangleq$ IF $req = C!NOT\_REQUEST$ THEN $\{\}$
ELSE $C!GetNonAdoptedGhostChildren(rf.id)$
$grf \triangleq C!ChooseGhost(ghosts)$
$grf\_updated \triangleq$ IF $grf = C!NOT\_FINISH$ THEN $C!NOT\_FINISH$
ELSE $[grf$ EXCEPT $!.isAdopted =$ TRUE$]$
$req\_updated \triangleq$ IF $req = C!NOT\_REQUEST$ THEN $C!NOT\_REQUEST$
ELSE $[req$ EXCEPT $!.markingDone =$ TRUE$]$
IN $\land\ req \neq C!NOT\_REQUEST$
$\land\ rf \neq C!NOT\_FINISH$
$\land$ IF $ghosts = \{\}$
THEN $\land\ rf\_set' = rf\_set$
$\land\ rec\_child' = (rec\_child \setminus \{req\}) \cup \{req\_updated\}$
ELSE $\land\ rf\_set' = (rf\_set \setminus \{grf\}) \cup \{grf\_updated\}$
$\land\ rec\_child' = rec\_child$
$\land\ step' = step + 1$

$\land$ UNCHANGED $\langle exec\_state,\ tasks,\ f\_set,\ lf\_set,\ msgs,$
$nxt\_finish\_id,\ nxt\_task\_id,\ nxt\_remote\_place,$
$killed,\ killed\_cnt,$
$lost\_tasks,\ lost\_f\_set,\ lost\_lf\_set,$
$rec\_to,\ rec\_from,\ rec\_from\_waiting\rangle$

$Store\_AddingGhostChildren \triangleq$
$\land\ exec\_state = \text{“running”}$
$\land$ LET $req \triangleq C!FindAddGhostChildrenRequest$
$rf \triangleq$ IF $req = C!NOT\_REQUEST$ THEN $C!NOT\_FINISH$
ELSE $C!FindResilientFinishById(req.finish\_id)$
$ghosts \triangleq C!GetAdoptedGhostChildren(rf.id)$
$rf\_updated \triangleq$ IF $req = C!NOT\_REQUEST$ THEN $C!NOT\_FINISH$
ELSE $[rf$ EXCEPT $!.ghost\_children = rf.ghost\_children \cup ghosts]$
IN $\land\ req \neq C!NOT\_REQUEST$
$\land\ rf \neq C!NOT\_FINISH$
$\land\ rf\_set' = (rf\_set \setminus \{rf\}) \cup \{rf\_updated\}$
$\land\ rec\_child' = rec\_child \setminus \{req\}$
$\land\ step' = step + 1$
$\land$ UNCHANGED $\langle exec\_state,\ tasks,\ f\_set,\ lf\_set,\ msgs,$
$nxt\_finish\_id,\ nxt\_task\_id,\ nxt\_remote\_place,$
$killed,\ killed\_cnt,$
$lost\_tasks,\ lost\_f\_set,\ lost\_lf\_set,$
$rec\_to,\ rec\_from,\ rec\_from\_waiting\rangle$

$Store\_CancellingTasksToDeadPlace \triangleq$
$\land\ exec\_state = \text{“running”}$
$\land$ LET $req \triangleq C!FindToDeadRequest$
$rf \triangleq$ IF $req = C!NOT\_REQUEST$ THEN $C!NOT\_FINISH$
ELSE $C!FindResilientFinishById(req.finish\_id)$
$rf\_updated \triangleq$ IF $req = C!NOT\_REQUEST$ THEN $C!NOT\_FINISH$
ELSE $[rf$ EXCEPT $!.transOrLive[req.from][req.to] = 0,$
$!.gc = rf.gc - rf.transOrLive[req.from][req.to]]$
IN $\land\ req \neq C!NOT\_REQUEST$
$\land\ rf \neq C!NOT\_FINISH$
$\land\ C!ApplyTerminateSignal2(rf,\ rf\_updated)$
$\land\ rec\_to' = rec\_to \setminus \{req\}$
$\land\ step' = step + 1$
$\land$ UNCHANGED $\langle exec\_state,\ tasks,\ f\_set,\ lf\_set,$
$nxt\_finish\_id,\ nxt\_task\_id,\ nxt\_remote\_place,$
$killed,\ killed\_cnt,$
$lost\_tasks,\ lost\_f\_set,\ lost\_lf\_set,$
$rec\_child,\ rec\_from,\ rec\_from\_waiting\rangle$

$Store\_SendingCountDroppedSignalToLocalFinish \triangleq$
$\land\ exec\_state = \text{“running”}$

```
  ∧ LET req ≜ C!FindFromDeadRequest
        rf ≜ IF req = C!NOT_REQUEST THEN C!NOT_FINISH
             ELSE IF ¬C!ResilientFinishExists(req.finish_id) THEN C!NOT_FINISH
             ELSE C!FindResilientFinishById(req.finish_id)
        msg ≜ IF req = C!NOT_REQUEST THEN C!NOT_MESSAGE
              ELSE [from ↦ "rf", to ↦ "dst", tag ↦ "countDropped",
                    finish_id ↦ rf.id,
                    src ↦ req.from, dst ↦ req.to,
                    num_sent ↦ rf.sent[req.from][req.to]]
    IN  ∧ req ≠ C!NOT_REQUEST
        ∧ rec_from' = rec_from \ {req}
        ∧ IF rf ≠ C!NOT_FINISH
          THEN ∧ C!SendMsg(msg)
               ∧ rec_from_waiting' = rec_from_waiting ∪ {req}
          ELSE ∧ msgs' = msgs  resilient finish has been released already
               ∧ rec_from_waiting' = rec_from_waiting
        ∧ step' = step + 1
  ∧ UNCHANGED ⟨exec_state, tasks, f_set, lf_set, rf_set,
               nxt_finish_id, nxt_task_id, nxt_remote_place,
               killed, killed_cnt,
               lost_tasks, lost_f_set, lost_lf_set,
               rec_child, rec_to⟩

Store_CancellingDroppedTasksFromDeadPlace ≜
  ∧ exec_state = "running"
  ∧ LET msg ≜ C!FindMessageToActivePlaceWithTag("rf", "countDroppedDone")
        from ≜ msg.src
        to ≜ msg.dst
        finish_id ≜ msg.finish_id
        req ≜ IF msg = C!NOT_MESSAGE THEN C!NOT_REQUEST
              ELSE C!FindFromDeadWaitingRequest(finish_id, from, to)
        rf ≜ IF msg = C!NOT_MESSAGE THEN C!NOT_FINISH
             ELSE IF ¬C!ResilientFinishExists(req.finish_id) THEN C!NOT_FINISH
             ELSE C!FindResilientFinishById(finish_id)
        rf_updated ≜ IF rf = C!NOT_FINISH THEN C!NOT_FINISH
                     ELSE [rf EXCEPT !.transOrLive[from][to] = rf.transOrLive[from][to] − msg.num_dr
                                     !.gc = rf.gc − msg.num_dropped]
    IN  ∧ msg ≠ C!NOT_MESSAGE
        ∧ rec_from_waiting' = rec_from_waiting \ {req}
        ∧ IF msg.num_dropped > 0
          THEN C!ApplyTerminateSignal(rf, rf_updated, msg)
          ELSE C!RecvCountDroppedResponse(msg)
        ∧ step' = step + 1
  ∧ UNCHANGED ⟨exec_state, tasks, f_set, lf_set,
               nxt_finish_id, nxt_task_id, nxt_remote_place,
```

$killed$, $killed\_cnt$,
$lost\_tasks$, $lost\_f\_set$, $lost\_lf\_set$,
rec_child, rec_to, $rec\_from\rangle$

$KillingPlace \triangleq$
$\land\ exec\_state = \text{“running”}$
$\land\ killed\_cnt < MAX\_KILL$
$\land$ LET $victim \triangleq$ CHOOSE $x \in (C!PlaceID \setminus killed\ ) : x \neq 0$
$victim\_tasks \triangleq C!FindLostTasks(victim)$
$victim\_finishes \triangleq C!FindLostFinishes(victim)$
$victim\_local\_finishes \triangleq C!FindLostLocalFinishes(victim)$
$rf\_to \triangleq C!FindImpactedResilientFinishToDead(victim)$
$rf\_from \triangleq C!FindImpactedResilientFinishFromDead(victim)$
IN $\land\ step \geq KILL\_FROM$
$\land\ step < KILL\_TO$
$\land\ killed' = killed \cup \{victim\}$
$\land\ killed\_cnt' = killed\_cnt + 1$
$\land\ lost\_tasks' = lost\_tasks \cup victim\_tasks$
$\land\ tasks' = tasks \setminus victim\_tasks$
$\land\ lost\_f\_set' = lost\_f\_set \cup victim\_finishes$
$\land\ f\_set' = f\_set \setminus victim\_finishes$
$\land\ lost\_lf\_set' = lost\_lf\_set \cup victim\_local\_finishes$
$\land\ lf\_set' = lf\_set \setminus victim\_local\_finishes$
$\land\ rec\_child' = rec\_child \cup \{$
$task \in C!GetChildrenTask :\ \land\ task.finish\_id \in rf\_to$
$\land\ task.victim = victim$
$\land\ task.markingDone =$ FALSE
$\}$
$\land\ rec\_to' = rec\_to \cup \{$
$task \in C!ConvTask : \exists\, rf \in rf\_set : \exists\, p \in C!PlaceID :$
$\land\ task.finish\_id = rf.id$
$\land\ task.finish\_id \in rf\_to$
$\land\ rf.transOrLive[p][victim] > 0$
$\land\ task.from = p$
$\land\ task.to = victim$
$\}$
$\land\ rec\_from' = rec\_from \cup \{$
$task \in C!ConvTask : \exists\, rf \in rf\_set : \exists\, p \in C!PlaceID :$
$\land\ task.finish\_id = rf.id$
$\land\ task.finish\_id \in rf\_to$
$\land\ rf.transOrLive[victim][p] > 0$
$\land\ task.to = p$
$\land\ task.from = victim$
$\}$

```
        ∧ step' = step + 1
  ∧ UNCHANGED ⟨exec_state, rf_set, msgs,
                nxt_finish_id, nxt_task_id, nxt_remote_place,
                rec_from_waiting⟩

Program_Terminating ≜
  ∧ exec_state = "running"
  ∧ LET root_task ≜ CHOOSE task ∈ tasks : task.id = C!ROOT_TASK_ID
        root_task_updated ≜ [root_task EXCEPT !.status = "terminated"]
    IN  ∧ root_task.status = "running"  root task unblocked
        ∧ tasks' = (tasks \ {root_task}) ∪ {root_task_updated}
        ∧ exec_state' = "success"
        ∧ step' = step + 1
  ∧ UNCHANGED ⟨f_set, lf_set, rf_set, msgs,
                nxt_finish_id, nxt_task_id, nxt_remote_place,
                killed, killed_cnt,
                lost_tasks, lost_f_set, lost_lf_set,
                rec_child, rec_to, rec_from, rec_from_waiting⟩
```

Possible next actions at each state

```
Next ≜
  ∨ Task_CreatingFinish
  ∨ Finish_CreatingRemoteTask
  ∨ Finish_TerminatingTask
  ∨ Finish_ReceivingPublishDoneSignal
  ∨ Finish_ReceivingReleaseSignal
  ∨ LocalFinish_CreatingRemoteTask
  ∨ LocalFinish_TerminatingTask
  ∨ LocalFinish_MarkingDeadPlace
  ∨ SendingTask
  ∨ DroppingTask
  ∨ ReceivingTask
  ∨ Store_ReceivingPublishSignal
  ∨ Store_ReceivingTransitSignal
  ∨ Store_ReceivingTerminateTaskSignal
  ∨ Store_ReceivingTerminateGhostSignal
  ∨ Store_FindingGhostChildren
  ∨ Store_AddingGhostChildren
  ∨ Store_CancellingTasksToDeadPlace
  ∨ Store_SendingCountDroppedSignalToLocalFinish
  ∨ Store_CancellingDroppedTasksFromDeadPlace
  ∨ KillingPlace
  ∨ Program_Terminating
```

We assume weak fairness on all actions (*i.e.* an action that remains forever enabled, must eventually be executed).

$$
\begin{aligned}
&\mathit{Liveness} \triangleq \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{Task\_CreatingFinish}) \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{Finish\_CreatingRemoteTask}) \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{Finish\_TerminatingTask}) \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{Finish\_ReceivingPublishDoneSignal}) \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{Finish\_ReceivingReleaseSignal}) \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{LocalFinish\_CreatingRemoteTask}) \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{LocalFinish\_TerminatingTask}) \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{LocalFinish\_MarkingDeadPlace}) \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{SendingTask}) \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{DroppingTask}) \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{ReceivingTask}) \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{Store\_ReceivingPublishSignal}) \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{Store\_ReceivingTransitSignal}) \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{Store\_ReceivingTerminateTaskSignal}) \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{Store\_ReceivingTerminateGhostSignal}) \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{Store\_FindingGhostChildren}) \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{Store\_AddingGhostChildren}) \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{Store\_CancellingTasksToDeadPlace}) \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{Store\_SendingCountDroppedSignalToLocalFinish}) \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{Store\_CancellingDroppedTasksFromDeadPlace}) \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{KillingPlace}) \\
&\quad \land \mathrm{WF}_{\mathit{Vars}}(\mathit{Program\_Terminating})
\end{aligned}
$$

Specification

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

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