

EXTENDS *FiniteSets, Integers, Sequences, TLC*

$Null \triangleq 0$
 $Cowns \triangleq 1 \dots 4$
 $MaxMessageCount \triangleq 4$
 $MaxMessageSize \triangleq 3$
 $OverloadThreshold \triangleq 2$
 $PriorityLevels \triangleq \{-1, 0, 1\}$
 $Pick(s) \triangleq \text{CHOOSE } x \in s : \text{TRUE}$
 $Min(s) \triangleq \text{CHOOSE } x \in s : \forall y \in s \setminus \{x\} : y > x$
 $Max(s) \triangleq \text{CHOOSE } x \in s : \forall y \in s \setminus \{x\} : y < x$
 $Range(f) \triangleq \{f[x] : x \in \text{DOMAIN } f\}$
 $Subsets(s, min, max) \triangleq$
 $\{x \in \text{SUBSET } s : (\text{Cardinality}(x) \geq min) \wedge (\text{Cardinality}(x) \leq max)\}$
 RECURSIVE $Concat(-)$
 $Concat(s) \triangleq \text{IF } s = \{\} \text{ THEN } \langle \rangle \text{ ELSE LET } x \triangleq Pick(s) \text{ IN } x \circ Concat(s \setminus \{x\})$

 VARIABLES *fuel, queue, scheduled, running, mutor, priority, blocker*
 vars $\triangleq \langle fuel, queue, scheduled, running, mutor, priority, blocker \rangle$

 $Messages \triangleq \text{UNION } \{Range(queue[c]) : c \in Cowns\}$

 $Normal(c) \triangleq priority[c] = 0$
 $Prioritized(c) \triangleq priority[c] = 1$
 $Muted(c) \triangleq priority[c] = -1$

 $EmptyQueue(c) \triangleq Len(queue[c]) = 0$
 $Overloaded(c) \triangleq Len(queue[c]) \geq OverloadThreshold$
 $Enqueue(c, m) \triangleq c :> Append(queue[c], m)$
 $Dequeue(c) \triangleq c :> Tail(queue[c])$

 RECURSIVE $Blockers(-)$
 $Blockers(c) \triangleq$
 $\text{IF } blocker[c] = Null \text{ THEN } \{\}$
 $\text{ELSE } \{blocker[c]\} \cup Blockers(blocker[c])$

 $Unblock(c) \triangleq [k \in \{c\} \cup Blockers(c) \mapsto Muted(k) \vee scheduled[k]]$

 $Running(c) \triangleq \exists k \in Cowns : running[k] \wedge c \in Head(queue[k])$

 $AcquiredBy(a, b) \triangleq$
 $\wedge a < b$
 $\wedge \exists m \in Range(queue[b]) : (a \in m) \wedge (b \in m)$
 $Acquired(c) \triangleq \exists k \in Cowns : AcquiredBy(c, k)$

 $MutedBy(a, b) \triangleq$

$$\begin{aligned}
& \wedge \text{Muted}(a) \\
& \wedge \exists m \in \text{Range}(\text{queue}[b]) : (b \notin m) \wedge (a \in m) \\
\text{Init} & \triangleq \\
& \wedge \text{fuel} = \text{MaxMessageCount} \\
& \wedge \text{queue} = [c \in \text{Cowns} \mapsto \langle \{c\} \rangle] \\
& \wedge \text{scheduled} = [c \in \text{Cowns} \mapsto \text{TRUE}] \\
& \wedge \text{running} = [c \in \text{Cowns} \mapsto \text{FALSE}] \\
& \wedge \text{mutor} = [c \in \text{Cowns} \mapsto \text{Null}] \\
& \wedge \text{priority} = [c \in \text{Cowns} \mapsto 0] \\
& \wedge \text{blocker} = [c \in \text{Cowns} \mapsto \text{Null}] \\
\text{Terminating} & \triangleq \\
& \wedge \forall c \in \text{Cowns} : \text{EmptyQueue}(c) \\
& \text{---} \\
& \wedge \text{UNCHANGED vars} \\
\text{ExternalReceive}(\text{cown}) & \triangleq \\
& \wedge \text{fuel} > 0 \\
& \text{---} \\
& \wedge \text{UNCHANGED} \langle \text{scheduled}, \text{running}, \text{mutor}, \text{priority}, \text{blocker} \rangle \\
& \wedge \text{fuel}' = \text{fuel} - 1 \\
& \# \text{ Receive a message from an external source} \\
& \wedge \exists \text{others} \in \text{Subsets}(\{c \in \text{Cowns} : c > \text{cown}\}, 0, \text{MaxMessageSize} - 1) : \\
& \quad \text{queue}' = \text{Enqueue}(\text{cown}, \{\text{cown}\} \cup \text{others}) @ @ \text{queue} \\
\text{Acquire}(\text{cown}) & \triangleq \\
& \wedge \text{scheduled}[\text{cown}] \\
& \wedge \neg \text{running}[\text{cown}] \\
& \wedge \neg \text{EmptyQueue}(\text{cown}) \\
& \wedge \text{cown} \in \text{Head}(\text{queue}[\text{cown}]) \\
& \wedge \text{cown} < \text{Max}(\text{Head}(\text{queue}[\text{cown}])) \\
& \text{---} \\
& \wedge \text{UNCHANGED} \langle \text{fuel}, \text{running}, \text{mutor} \rangle \\
& \# \text{ Unschedule and forward the message to the next cown.} \\
& \wedge \text{LET} \\
& \quad \text{msg} \triangleq \text{Head}(\text{queue}[\text{cown}]) \\
& \quad \text{next} \triangleq \text{Min}(\{c \in \text{msg} : c > \text{cown}\}) \\
& \text{IN} \\
& \wedge \text{queue}' = \text{Enqueue}(\text{next}, \text{msg}) @ @ \text{Dequeue}(\text{cown}) @ @ \text{queue} \\
& \wedge \text{blocker}' = (\text{cown} :> \text{next}) @ @ \text{blocker} \\
& \# \text{ Prioritize this cown and next if either are prioritized.} \\
& \wedge \text{IF } \exists c \in \{\text{cown}, \text{next}\} : \text{Prioritized}(c) \text{ THEN} \\
& \quad \text{Unblock next.} \\
& \wedge \text{priority}' = (\text{next} :> 1) @ @ [c \in \text{Blockers}(\text{next}) \mapsto 1] @ @ \text{priority} \\
& \wedge \text{scheduled}' = \text{Unblock}(\text{next}) @ @ (\text{cown} :> \text{FALSE}) @ @ \text{scheduled}
\end{aligned}$$

```

ELSE
  ∧ UNCHANGED ⟨priority⟩
  ∧ scheduled' = (cown :> FALSE) @@ scheduled

Unmute(cown) ≜
  ∧ scheduled[cown]
  ∧ ¬running[cown]
  ∧ ¬EmptyQueue(cown)
  ∧ cown ∉ Head(queue[cown])
  ———
  ∧ UNCHANGED ⟨fuel, running, mutor, blocker⟩
  # Remove message from queue.
  ∧ queue' = Dequeue(cown) @@ queue
  # Reschedule muted cowns.
  ∧ LET muted ≜ {c ∈ Head(queue[cown]) : Muted(c)} IN
    ∧ priority' = [c ∈ muted ↦ 0] @@ priority
    ∧ scheduled' = [c ∈ muted ↦ TRUE] @@ scheduled

PreRun(cown) ≜
  ∧ scheduled[cown]
  ∧ ¬running[cown]
  ∧ ¬EmptyQueue(cown)
  ∧ cown = Max(Head(queue[cown]))
  ———
  ∧ UNCHANGED ⟨fuel, queue, scheduled, mutor, priority⟩
  # Set max cown in current message to running
  ∧ running' = (cown :> TRUE) @@ running
  ∧ blocker' = [c ∈ Head(queue[cown]) ↦ Null] @@ blocker

Send(cown) ≜
  ∧ running[cown]
  ∧ fuel > 0
  # ———
  ∧ UNCHANGED ⟨running, blocker⟩
  ∧ fuel' = fuel − 1
  ∧ ∃ receivers ∈ Subsets(Cowns, 1, MaxMessageSize) :
    LET
      next ≜ Min(receivers)
      senders ≜ Head(queue[cown])
      mutors ≜ {c ∈ receivers : Overloaded(c)}
    IN
      # Place message for receivers in the first receiver's queue.
      ∧ queue' = Enqueue(next, senders) @@ queue
      ∧ IF (∃ c ∈ receivers : Prioritized(c) ∨ Overloaded(next)) THEN
        # Unblock next.
        ∧ priority' = (next :> 1) @@ [c ∈ Blockers(next) ↦ 1] @@ priority

```

```

 $\wedge \text{scheduled}' = \text{Unblock}(\text{next}) @ @ \text{scheduled}$ 
# Set mutor if any receiver is overloaded and there are no receivers in the set of senders.
 $\wedge$  IF
 $\wedge \text{mutors} \neq \{\}$ 
 $\wedge \text{mutor}[\text{cown}] = \text{Null}$ 
 $\wedge (\text{senders} \cap \text{receivers}) = \{\}$ 
# The priority of senders is checked before muting in PostRun.
THEN  $\text{mutor}' = (\text{cown} :> \text{Min}(\text{mutors})) @ @ \text{mutor}$ 
ELSE UNCHANGED  $\langle \text{mutor} \rangle$ 
ELSE
UNCHANGED  $\langle \text{scheduled}, \text{mutor}, \text{priority} \rangle$ 

PostRun(cown)  $\triangleq$ 
 $\wedge \text{running}[\text{cown}]$ 
———
 $\wedge$  UNCHANGED  $\langle \text{fuel}, \text{blocker} \rangle$ 
 $\wedge \text{running}' = (\text{cown} :> \text{FALSE}) @ @ \text{running}$ 
 $\wedge \text{mutor}' = (\text{cown} :> \text{Null}) @ @ \text{mutor}$ 
 $\wedge \text{LET } \text{msg} \triangleq \text{Head}(\text{queue}[\text{cown}]) \text{IN}$ 
# Mute if mutor is set and no running cowns are overloaded.
 $\wedge$  IF
 $\wedge \text{mutor}[\text{cown}] \neq \text{Null}$ 
 $\wedge \forall c \in \text{msg} : \neg \text{Overloaded}(c) \wedge \neg \text{Prioritized}(c)$ 
THEN
 $\wedge \text{priority}' = [c \in \text{msg} \mapsto -1] @ @ \text{priority}$ 
 $\wedge \text{scheduled}' = [c \in \text{msg} \mapsto \text{FALSE}] @ @ \text{scheduled}$ 
# Send unmute message to mutor
 $\wedge \text{queue}' = \text{Enqueue}(\text{mutor}[\text{cown}], \text{msg}) @ @ \text{Dequeue}(\text{cown}) @ @ \text{queue}$ 
ELSE
UNCHANGED  $\langle \text{priority} \rangle$ 
 $\wedge \text{scheduled}' = [c \in \text{msg} \mapsto \text{TRUE}] @ @ \text{scheduled}$ 
 $\wedge \text{queue}' = \text{Dequeue}(\text{cown}) @ @ \text{queue}$ 

RunStep(cown)  $\triangleq$ 
 $\vee \text{ExternalReceive}(\text{cown}) \setminus * \#$  Very expensive check
 $\vee \text{Acquire}(\text{cown})$ 
 $\vee \text{Unmute}(\text{cown})$ 
 $\vee \text{PreRun}(\text{cown})$ 
 $\vee \text{Send}(\text{cown})$ 
 $\vee \text{PostRun}(\text{cown})$ 

Next  $\triangleq \exists c \in \text{Cowns} : \text{RunStep}(c)$ 

Spec  $\triangleq$ 
 $\wedge \text{Init}$ 
 $\wedge \Box [\text{Next} \vee \text{Terminating}]_{\text{vars}}$ 

```

$\wedge \forall c \in \text{Cowns} : \text{WF}_{\text{vars}}(\text{RunStep}(c))$

Properties

Ensure that the termination condition is reached by the model.

$\text{Termination} \triangleq \Diamond \Box (\forall c \in \text{Cowns} : \text{EmptyQueue}(c))$

Invariants

Ensure that the model produces finite messages.

$\text{MessageLimit} \triangleq \text{Cardinality}(\text{Messages}) \leq (\text{Cardinality}(\text{Cowns}) + \text{MaxMessageCount})$

Cowns are acquired by one running message at a time.

$\text{UniqueAcquisition} \triangleq$
 $\text{LET } \text{msgs} \triangleq \text{Concat}(\{\langle \text{Head}(\text{queue}[c]) \rangle : c \in \{k \in \text{Cowns} : \text{running}[k]\}\})$
 $\text{IN } \text{Cardinality}(\text{Range}(\text{msgs})) = \text{Len}(\text{msgs})$

TODO: Token messages?

Each message has at least one cown.

$\text{NoTokens} \triangleq \forall c \in \text{Cowns} : \text{Len}(\text{SelectSeq}(\text{queue}[c], \text{LAMBDA } m : m = \{\})) = 0$

A running cown must be scheduled and be the max cown in the message at the head of its queue.

$\text{RunningImplication} \triangleq \forall c \in \text{Cowns} : \text{running}[c] \Rightarrow$
 $\wedge \text{scheduled}[c]$
 $\wedge c = \text{Max}(\text{Head}(\text{queue}[c]))$
 $\wedge \forall k \in \text{Head}(\text{queue}[c]) : (k < c) \Rightarrow \text{AcquiredBy}(k, c)$

An acquired cown is not scheduled.

$\text{AcquiredImplication} \triangleq \forall c \in \text{Cowns} : \text{Acquired}(c) \Rightarrow$
 $\wedge \neg \text{scheduled}[c]$

A muted cown is not scheduled or running.

$\text{MutedImplication} \triangleq \forall c \in \text{Cowns} : \text{Muted}(c) \equiv$
 $\wedge \exists k \in \text{Cowns} : \text{MutedBy}(c, k)$
 $\wedge \neg \text{scheduled}[c]$
 $\wedge \neg \text{Running}(c)$

A muted cown exists in an unmute message in the queue of at least one mutor.

$\text{MutedInUnmuteMsg} \triangleq$
 $\forall m \in \{c \in \text{Cowns} : \text{Muted}(c)\} :$
 $\text{Cardinality}(\{c \in \text{Cowns} : \text{MutedBy}(m, c)\}) > 0$

A cown may be acquired by at most one message.

$\text{AcquiredOnce} \triangleq$
 $\forall a \in \{c \in \text{Cowns} : \text{Acquired}(c)\} :$
 $\text{Cardinality}(\{c \in \text{Cowns} : \text{AcquiredBy}(a, c)\}) = 1$

An acquired cown is either acquired by a cown in its blocker set or it is running.

$\text{AcquiredByBlocker} \triangleq \forall \langle a, b \rangle \in \text{Cowns} \times \text{Cowns} :$

$AcquiredBy(a, b) \Rightarrow b \in Blockers(a) \vee Running(a)$

A prioritized *cown* is not acquired by a muted *cown*.

$PrioritizedNotAcquiredByMuted \triangleq \forall \langle o, m \rangle \in Cowns \times Cowns :$
 $Prioritized(o) \wedge Muted(m) \Rightarrow \neg AcquiredBy(o, m)$

(*

\ * <https://github.com/tlaplus/Examples/blob/master/specifications/TransitiveClosure/TransitiveClosure.tla#L114>

$TC(R) \triangleq$

LET

$S \triangleq \{r[1] : r \in R\} \cup \{r[2] : r \in R\}$

RECURSIVE $TCR(_)$

$TCR(T) \triangleq$

IF $T = \{\}$ THEN R

ELSE

LET

$r \triangleq \text{CHOOSE } s \in T : \text{TRUE}$

$RR \triangleq TCR(T \setminus \{r\})$

IN

$RR \cup \{\langle s, t \rangle \in S \times S : \langle s, r \rangle \in RR \wedge \langle r, t \rangle \in RR\}$

IN

$TCR(S)$

$CyclicTransitiveClosure(R(-, -)) \triangleq$

LET $s \triangleq \{\langle a, b \rangle \in Cowns \times Cowns : R(a, b)\}$

IN $\exists c \in Cowns : \langle c, c \rangle \in TC(s)$

*)