

EXTENDS *FiniteSets, Integers, Sequences, TLC*

$Null \triangleq 0$   
 $Cowns \triangleq 1..3 \# \text{ TODO: } 4$   
 $MaxMessageCount \triangleq 4 \# \text{ TODO: } 4$   
 $MaxMessageSize \triangleq 3$   
 $OverloadThreshold \triangleq 2$   
 $PriorityLevels \triangleq \{-1, 0, 1, 2\}$   
 $Pick(s) \triangleq \text{CHOOSE } x \in s : \text{TRUE}$   
 $Min(s) \triangleq \text{CHOOSE } x \in s : \forall y \in s \setminus \{x\} : y > x$   
 $Max(s) \triangleq \text{CHOOSE } x \in s : \forall y \in s \setminus \{x\} : y < x$   
 $Range(f) \triangleq \{f[x] : x \in \text{DOMAIN } f\}$   
 $Subsets(s, min, max) \triangleq$   
 $\{x \in \text{SUBSET } s : (\text{Cardinality}(x) \geq min) \wedge (\text{Cardinality}(x) \leq max)\}$   
 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, muted, blocker*  
 vars  $\triangleq \langle fuel, queue, scheduled, running, mutor, muted, blocker \rangle$   
  
 $Messages \triangleq \text{UNION } \{Range(queue[c]) : c \in Cowns\}$   
 $EmptyQueue(c) \triangleq Len(queue[c]) = 0$   
 $Overloaded(c) \triangleq Len(queue[c]) \geq OverloadThreshold$   
 $Enqueue(c, m) \triangleq c :> Append(queue[c], m)$   
 $Dequeue(c) \triangleq c :> Tail(queue[c])$   
  
 RECURSIVE  $Blockers(-)$   
 $Blockers(c) \triangleq$   
 $\text{IF } blocker[c] = Null \text{ THEN } \{\}$   
 $\text{ELSE } \{blocker[c]\} \cup Blockers(blocker[c])$   
  
 $Running(c) \triangleq \exists k \in Cowns : running[k] \wedge c \in Head(queue[k])$   
  
 $AcquiredBy(a, b) \triangleq$   
 $\wedge a < b$   
 $\wedge \exists c \in Cowns : a \in \text{UNION } Range(queue[b])$   
 $\wedge b = Min(\{c \in Cowns : a \in \text{UNION } Range(queue[b])\})$   
 $Acquired(c) \triangleq \exists k \in Cowns : AcquiredBy(c, k)$   
  
 $MutedBy(a, b) \triangleq$   
 $\wedge muted[a]$   
 $\wedge \exists m \in Range(queue[b]) : (b \notin m) \wedge (a \in m)$   
  
 $Init \triangleq$   
 $\wedge fuel = MaxMessageCount$

$$\begin{aligned}
&\wedge \text{queue} = [c \in \text{Cowns} \mapsto \langle \{c\} \rangle] \\
&\wedge \text{scheduled} = [c \in \text{Cowns} \mapsto \text{TRUE}] \\
&\wedge \text{running} = [c \in \text{Cowns} \mapsto \text{FALSE}] \\
&\wedge \text{mutor} = [c \in \text{Cowns} \mapsto \text{Null}] \\
&\wedge \text{muted} = [c \in \text{Cowns} \mapsto \text{FALSE}] \\
&\wedge \text{blocker} = [c \in \text{Cowns} \mapsto \text{Null}]
\end{aligned}$$

$$\begin{aligned}
&\text{Terminating} \triangleq \\
&\wedge \forall c \in \text{Cowns} : \text{EmptyQueue}(c)
\end{aligned}$$


---


$$\wedge \text{UNCHANGED vars}$$

$$\begin{aligned}
&\text{ExternalReceive}(cown) \triangleq \\
&\wedge \text{fuel} > 0 \\
&\quad \text{UNCHANGED } \langle \text{scheduled}, \text{running}, \text{mutor}, \text{muted}, \text{blocker} \rangle \\
&\quad \wedge \text{fuel}' = \text{fuel} - 1 \\
&\quad \# \text{ Receive a message from an external source} \\
&\quad \wedge \exists \text{others} \in \text{Subsets}(\{c \in \text{Cowns} : c > cown\}, 0, \text{MaxMessageSize} - 1) : \\
&\quad \quad \text{queue}' = \text{Enqueue}(cown, \{cown\} \cup \text{others}) @ @ \text{queue}
\end{aligned}$$

$$\begin{aligned}
&\text{Acquire}(cown) \triangleq \\
&\quad \wedge \text{scheduled}[cown] \\
&\quad \wedge \neg \text{running}[cown] \\
&\quad \wedge \neg \text{EmptyQueue}(cown) \\
&\quad \wedge cown \in \text{Head}(\text{queue}[cown]) \\
&\quad \wedge cown < \text{Max}(\text{Head}(\text{queue}[cown])) \\
&\quad \text{UNCHANGED } \langle \text{fuel}, \text{running}, \text{mutor} \rangle \\
&\quad \# \text{ Unschedule and forward the message to the next } cown. \\
&\quad \wedge \text{LET} \\
&\quad \quad \text{msg} \triangleq \text{Head}(\text{queue}[cown]) \\
&\quad \quad \text{next} \triangleq \text{Min}(\{c \in \text{msg} : c > cown\}) \\
&\quad \text{IN} \\
&\quad \wedge \text{queue}' = \text{Enqueue}(\text{next}, \text{msg}) @ @ \text{Dequeue}(cown) @ @ \text{queue} \\
&\quad \wedge \text{blocker}' = (cown :> \text{next}) @ @ \text{blocker} \\
&\quad \wedge \text{IF } (\text{Overloaded}(cown) \vee \text{Overloaded}(\text{next})) \wedge \text{muted}[\text{next}] \text{ THEN} \\
&\quad \quad \wedge \text{scheduled}' = (\text{next} :> \text{TRUE}) @ @ (\text{cown} :> \text{FALSE}) @ @ \text{scheduled} \\
&\quad \quad \wedge \text{muted}' = (\text{next} :> \text{FALSE}) @ @ \text{muted} \\
&\quad \text{ELSE} \\
&\quad \quad \wedge \text{UNCHANGED } \langle \text{muted} \rangle \\
&\quad \quad \wedge \text{scheduled}' = (\text{cown} :> \text{FALSE}) @ @ \text{scheduled}
\end{aligned}$$

$$\begin{aligned}
&\text{Unmute}(cown) \triangleq \\
&\quad \wedge \text{scheduled}[cown] \\
&\quad \wedge \neg \text{running}[cown]
\end{aligned}$$

$$\begin{array}{l}
\wedge \neg \text{EmptyQueue}(\text{cown}) \\
\wedge \text{cown} \notin \text{Head}(\text{queue}[\text{cown}]) \\
\hline
\wedge \text{UNCHANGED } \langle \text{fuel}, \text{running}, \text{mutor}, \text{blocker} \rangle \\
\# \text{ Remove message from queue.} \\
\wedge \text{queue}' = \text{Dequeue}(\text{cown}) @ @ \text{queue} \\
\# \text{ Reschedule muted cowns.} \\
\wedge \text{muted}' = [c \in \text{Head}(\text{queue}[\text{cown}]) \mapsto \text{FALSE}] @ @ \text{muted} \\
\wedge \text{scheduled}' = [c \in \text{Head}(\text{queue}[\text{cown}]) \mapsto \text{TRUE}] @ @ \text{scheduled} \\
\\
\text{PreRun}(\text{cown}) \triangleq \\
\wedge \text{scheduled}[\text{cown}] \\
\wedge \neg \text{running}[\text{cown}] \\
\wedge \neg \text{EmptyQueue}(\text{cown}) \\
\wedge \text{cown} = \text{Max}(\text{Head}(\text{queue}[\text{cown}])) \\
\hline
\wedge \text{UNCHANGED } \langle \text{fuel}, \text{queue}, \text{scheduled}, \text{mutor}, \text{muted} \rangle \\
\# \text{ Set max cown in current message to running} \\
\wedge \text{running}' = (\text{cown} :> \text{TRUE}) @ @ \text{running} \\
\wedge \text{blocker}' = [c \in \text{Head}(\text{queue}[\text{cown}]) \mapsto \text{Null}] @ @ \text{blocker} \\
\\
\text{Send}(\text{cown}) \triangleq \\
\wedge \text{running}[\text{cown}] \\
\wedge \text{fuel} > 0 \\
\hline
\wedge \text{UNCHANGED } \langle \text{running}, \text{blocker} \rangle \\
\wedge \text{fuel}' = \text{fuel} - 1 \\
\# \text{ Select set of receivers} \\
\wedge \exists \text{receivers} \in \text{Subsets}(\text{Cowns}, 1, \text{MaxMessageSize}) : \\
\text{LET} \\
\quad \text{next} \triangleq \text{Min}(\text{receivers}) \\
\quad \text{senders} \triangleq \text{Head}(\text{queue}[\text{cown}]) \\
\quad \text{mutors} \triangleq \{c \in \text{receivers} : \text{Overloaded}(c)\} \\
\text{IN} \\
\quad \# \text{ Place message for receivers in the first receiver's queue.} \\
\quad \wedge \text{queue}' = \text{Enqueue}(\text{next}, \text{receivers}) @ @ \text{queue} \\
\quad \wedge \text{IF } (\text{Overloaded}(\text{cown}) \vee \text{Overloaded}(\text{next})) \wedge \text{muted}[\text{next}] \text{ THEN} \\
\quad \quad \wedge \text{UNCHANGED } \langle \text{mutor} \rangle \\
\quad \quad \# \text{ Unmute next.} \\
\quad \quad \wedge \text{scheduled}' = (\text{next} :> \text{TRUE}) @ @ \text{scheduled} \\
\quad \quad \wedge \text{muted}' = (\text{next} :> \text{FALSE}) @ @ \text{muted} \\
\quad \text{ELSE} \\
\quad \quad \wedge \text{UNCHANGED } \langle \text{scheduled}, \text{muted} \rangle \\
\quad \quad \# \text{ Set mutor if any receiver is overloaded and there are no receivers in the set of senders.} \\
\quad \quad \wedge \text{IF}
\end{array}$$

$\wedge mutors \neq \{\}$   
 $\wedge mutor[cown] = Null$   
 $\wedge (senders \cap receivers) = \{\}$   
 THEN  $mutor' = (cown :> Min(mutors)) @@ mutor$   
 ELSE UNCHANGED  $\langle mutor \rangle$

$PostRun(cown) \triangleq$   
 $\wedge running[cown]$   


---

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

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

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

$Spec \triangleq$   
 $\wedge Init$   
 $\wedge \Box [Next \vee Terminating]_{vars}$   
 $\wedge \forall c \in Cowns : WF_{vars}(RunStep(c))$

$\# \text{ Properties}$

$\# \text{ Ensure that the termination condition is reached by the model.}$

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

$\# \text{ Invariants}$

$\# \text{ Ensure that the model produces finite messages.}$

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

# *Cowns* are acquired by one running message at a time.

$UniqueAcquisition \triangleq$

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

# Each queue has at most one token message.

$LoneToken \triangleq \forall c \in Cowns : Len(SelectSeq(queue[c], LAMBDA m : m = \{\})) \leq 1$

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

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

# An acquired *cown* is not scheduled.

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

# A muted *cown* is not scheduled or running.

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

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

$MutedOnce \triangleq$   
 $\forall m \in \{c \in Cowns : muted[c]\} :$   
 $Cardinality(\{c \in Cowns : MutedBy(m, c)\}) > 0$

# A *cown* may only be acquired by one message.

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

# An acquired *cown* is acquired by a *cown* in its blocker set.

$AcquiredByBlocker \triangleq \forall \langle a, b \rangle \in Cowns \times Cowns :$   
 $AcquiredBy(a, b) \Rightarrow b \in Blockers(a)$

# An overloaded *cown* doesn't exist in a muted *cown*'s queue.

$OverloadedNotInMutedQueue \triangleq \forall \langle o, m \rangle \in Cowns \times Cowns :$   
 $Overloaded(o) \wedge muted[m] \Rightarrow o \notin \text{UNION } Range(queue[m])$

(\*

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

$$\begin{aligned}
TC(R) &\triangleq \\
&\text{LET} \\
&\quad S \triangleq \{r[1] : r \in R\} \cup \{r[2] : r \in R\} \\
&\quad \text{RECURSIVE } TCR(-) \\
&\quad TCR(T) \triangleq \\
&\quad \text{IF } T = \{\} \text{ THEN } R \\
&\quad \text{ELSE} \\
&\quad \quad \text{LET} \\
&\quad \quad \quad r \triangleq \text{CHOOSE } s \in T : \text{TRUE} \\
&\quad \quad \quad RR \triangleq TCR(T \setminus \{r\}) \\
&\quad \text{IN} \\
&\quad \quad RR \cup \{\langle s, t \rangle \in S \times S : \langle s, r \rangle \in RR \wedge \langle r, t \rangle \in RR\} \\
&\text{IN} \\
&\quad TCR(S) \\
CyclicTransitiveClosure(R(-, -)) &\triangleq \\
&\text{LET } s \triangleq \{\langle a, b \rangle \in Cowns \times Cowns : R(a, b)\} \\
&\text{IN } \exists c \in Cowns : \langle c, c \rangle \in TC(s) \\
&*)
\end{aligned}$$