

EXTENDS *FiniteSets, Integers, Sequences, TLC*

Null $\triangleq 0$

Cowns $\triangleq 1 \dots 4$

BehaviourLimit $\triangleq 4$

OverloadThreshold $\triangleq 2$

PriorityLevels $\triangleq \{-1, 0, 1\}$

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*) $\triangleq \{f[x] : x \in \text{DOMAIN } f\}$

Pick(*s*) $\triangleq \text{CHOOSE } x \in s : \text{TRUE}$

ReduceSet(*op*(*_, _*), *set*, *acc*) \triangleq

LET *f*[*s* \in SUBSET *set*] \triangleq

IF *s* = {} THEN *acc* ELSE LET *x* \triangleq *Pick*(*s*) IN *op*(*x*, *f*[*s* \ {*x*}])

IN *f*[*set*]

VARIABLES *fuel*, *queue*, *scheduled*, *running*, *priority*, *blocker*, *mutor*, *mute*

vars $\triangleq \langle \textit{fuel}, \textit{queue}, \textit{scheduled}, \textit{running}, \textit{priority}, \textit{blocker}, \textit{mutor}, \textit{mute} \rangle$

Sleeping(*c*) $\triangleq \textit{scheduled}[c] \wedge (\textit{Len}(\textit{queue}[c]) = 0)$

Available(*c*) $\triangleq \textit{scheduled}[c] \wedge (\textit{Len}(\textit{queue}[c]) > 0)$

Overloaded(*c*) $\triangleq \textit{Len}(\textit{queue}[c]) > \textit{OverloadThreshold}$

Muted(*c*) $\triangleq c \in \text{UNION } \textit{Range}(\textit{mute})$

CurrentMessage(*c*) $\triangleq \text{IF } \textit{Len}(\textit{queue}[c]) > 0 \text{ THEN } \textit{Head}(\textit{queue}[c]) \text{ ELSE } \{\}$

LowPriority(*cs*) $\triangleq \{c \in cs : \textit{priority}[c] = -1\}$

HighPriority(*cs*) $\triangleq \{c \in cs : \textit{priority}[c] = 1\}$

RequiresPriority(*c*) \triangleq

$\vee \textit{Overloaded}(c)$

$\vee \exists m \in \textit{Range}(\textit{queue}[c]) : \exists k \in m \setminus \{c\} : \textit{priority}[k] = 1$

RECURSIVE *Blockers*(*_*)

Blockers(*c*) \triangleq

IF *blocker*[*c*] = *Null* THEN {} ELSE {*blocker*[*c*]} \cup *Blockers*(*blocker*[*c*])

Prioritizing(*cs*) \triangleq

LET *unprioritized* $\triangleq \{c \in cs : \textit{priority}[c] < 1\}$ IN

unprioritized \cup UNION {*Blockers*(*c*) : *c* \in *unprioritized*}

$$\begin{aligned}
& \text{ValidMutor}(c) \triangleq \\
& \quad \vee (\text{priority}[c] = 1) \wedge \text{Overloaded}(c) \\
& \quad \vee (\text{priority}[c] = -1) \\
& \text{Init} \triangleq \\
& \quad \wedge \text{fuel} = \text{BehaviourLimit} \\
& \quad \wedge \text{queue} = [c \in \text{Cowns} \mapsto \langle \{c\} \rangle] \\
& \quad \wedge \text{scheduled} = [c \in \text{Cowns} \mapsto \text{TRUE}] \\
& \quad \wedge \text{running} = [c \in \text{Cowns} \mapsto \text{FALSE}] \\
& \quad \wedge \text{priority} = [c \in \text{Cowns} \mapsto 0] \\
& \quad \wedge \text{blocker} = [c \in \text{Cowns} \mapsto \text{Null}] \\
& \quad \wedge \text{mutor} = [c \in \text{Cowns} \mapsto \text{Null}] \\
& \quad \wedge \text{mute} = [c \in \text{Cowns} \mapsto \{\}] \\
& \text{Terminating} \triangleq \\
& \quad \wedge \forall c \in \text{Cowns} : \text{Sleeping}(c) \\
& \quad \wedge \text{UNCHANGED vars} \\
& \text{Acquire}(cown) \triangleq \\
& \quad \text{LET } msg \triangleq \text{CurrentMessage}(cown) \text{ IN} \\
& \quad \wedge \text{Available}(cown) \\
& \quad \wedge cown < \text{Max}(msg) \\
& \quad \wedge \text{IF } \exists c \in msg : \text{priority}[c] = 1 \text{ THEN} \\
& \quad \quad \text{LET } prioritizing \triangleq \text{Prioritizing}(\{c \in msg : c > cown\}) \text{ IN} \\
& \quad \quad \text{LET } unmuting \triangleq \text{LowPriority}(prioritizing) \text{ IN} \\
& \quad \quad \wedge \text{priority}' = [c \in prioritizing \mapsto 1] @@ \text{priority} \\
& \quad \quad \wedge \text{scheduled}' = (cown :> \text{FALSE}) @@ [c \in unmuting \mapsto \text{TRUE}] @@ \text{scheduled} \\
& \quad \text{ELSE} \\
& \quad \quad \wedge \text{scheduled}' = (cown :> \text{FALSE}) @@ \text{scheduled} \\
& \quad \quad \wedge \text{UNCHANGED } \langle \text{priority}, \text{mute} \rangle \\
& \quad \wedge \text{LET } next \triangleq \text{Min}(\{c \in msg : c > cown\}) \text{ IN} \\
& \quad \quad \wedge \text{blocker}' = (cown :> next) @@ \text{blocker} \\
& \quad \quad \wedge \text{LET } q \triangleq (cown :> \text{Tail}(\text{queue}[cown])) @@ \text{queue} \text{ IN} \\
& \quad \quad \quad \text{queue}' = (next :> \text{Append}(\text{queue}[next], msg)) @@ q \\
& \quad \wedge \text{UNCHANGED } \langle \text{fuel}, \text{running}, \text{mutor}, \text{mute} \rangle \\
& \text{Prerun}(cown) \triangleq \\
& \quad \text{LET } msg \triangleq \text{CurrentMessage}(cown) \text{ IN} \\
& \quad \wedge \text{scheduled}[cown] \\
& \quad \wedge \neg \text{running}[cown] \\
& \quad \wedge \text{IF } msg = \{\} \text{ THEN FALSE ELSE } cown = \text{Max}(msg) \\
& \quad \wedge \text{priority}' = (cown :> \text{IF } \text{RequiresPriority}(cown) \text{ THEN } 1 \text{ ELSE } 0) @@ \text{priority} \\
& \quad \wedge \text{running}' = (cown :> \text{TRUE}) @@ \text{running} \\
& \quad \wedge \text{blocker}' = [c \in msg \mapsto \text{Null}] @@ \text{blocker} \\
& \quad \wedge \text{UNCHANGED } \langle \text{fuel}, \text{queue}, \text{scheduled}, \text{mutor}, \text{mute} \rangle
\end{aligned}$$

$Send(cown) \triangleq$
 LET $senders \triangleq CurrentMessage(cown)$ IN
 $\wedge running[cown]$
 $\wedge fuel > 0$
 $\wedge \exists receivers \in SUBSET\ Cowns :$
 $\wedge Cardinality(receivers) > 0$
 $\wedge queue' =$
 $(Min(receivers) :> Append(queue[Min(receivers)], receivers)) @@ queue$
 \wedge IF $\exists c \in receivers : priority[c] = 1$ THEN
 LET $prioritizing \triangleq Prioritizing(receivers)$ IN
 LET $unmuting \triangleq LowPriority(prioritizing)$ IN
 $\wedge priority' = [c \in prioritizing \mapsto 1] @@ priority$
 $\wedge scheduled' = [c \in unmuting \mapsto TRUE] @@ scheduled$
 \wedge LET $mutors \triangleq \{c \in receivers \setminus senders : ValidMutor(c)\}$ IN
 IF
 $\wedge mutors \neq \{\}$
 $\wedge mutor[cown] = Null$
 $\wedge \forall c \in senders : priority[c] = 0$
 $\wedge \forall c \in senders : c \notin receivers\ \text{TODO: justify}$
 THEN
 $\wedge mutor' = (cown :> Min(mutors)) @@ mutor$
 ELSE
 $\wedge UNCHANGED \langle mutor \rangle$
 ELSE
 $\wedge UNCHANGED \langle scheduled, priority, mutor \rangle$
 $\wedge fuel' = fuel - 1$
 $\wedge UNCHANGED \langle running, blocker, mute \rangle$
 $Complete(cown) \triangleq$
 LET $msg \triangleq CurrentMessage(cown)$ IN
 $\wedge running[cown]$
 \wedge IF $mutor[cown] \neq Null$ THEN
 LET $muting \triangleq \{c \in msg : priority[c] = 0\}$ IN
 $\wedge priority' = [c \in muting \mapsto -1] @@ priority$
 $\wedge mute' = (mutor[cown] :> mute[mutor[cown]] \cup muting) @@ mute$
 $\wedge scheduled' = [c \in msg \mapsto c \notin muting] @@ scheduled$
 ELSE
 $\wedge scheduled' = [c \in msg \mapsto TRUE] @@ scheduled$
 $\wedge priority' =$
 $(cown :> \text{IF } Len(queue[cown]) = 1 \text{ THEN } 0 \text{ ELSE } priority[cown]) @@$
 $[c \in msg \setminus \{cown\} \mapsto \text{IF } Len(queue[c]) = 0 \text{ THEN } 0 \text{ ELSE } priority[c]] @@$
 $priority$
 $\wedge UNCHANGED \langle mute \rangle$
 $\wedge queue' = (cown :> Tail(queue[cown])) @@ queue$
 $\wedge running' = (cown :> FALSE) @@ running$

$$\wedge \text{mutor}' = (\text{cown} := \text{Null}) @@ \text{mutor} \\ \wedge \text{UNCHANGED } \langle \text{fuel}, \text{blocker} \rangle$$

$$\begin{aligned} \text{Unmute} &\triangleq \\ &\text{LET } \text{invalid_keys} \triangleq \{c \in \text{DOMAIN } \text{mute} : \text{priority}[c] = 0\} \text{IN} \\ &\text{LET } \text{unmuting} \triangleq \text{UNION } \text{Range}([k \in \text{invalid_keys} \mapsto \text{LowPriority}(\text{mute}[k])]) \text{IN} \\ &\wedge \text{unmuting} \neq \{\} \\ &\wedge \text{priority}' = [c \in \text{unmuting} \mapsto 0] @@ \text{priority} \\ &\wedge \text{mute}' = [c \in \text{invalid_keys} \mapsto \{\}] @@ \text{mute} \\ &\wedge \text{scheduled}' = [c \in \text{unmuting} \mapsto \text{TRUE}] @@ \text{scheduled} \\ &\wedge \text{UNCHANGED } \langle \text{fuel}, \text{queue}, \text{running}, \text{blocker}, \text{mutor} \rangle \end{aligned}$$

$$\begin{aligned} \text{Run}(\text{cown}) &\triangleq \\ &\vee \text{Acquire}(\text{cown}) \\ &\vee \text{Prerun}(\text{cown}) \\ &\vee \text{Send}(\text{cown}) \\ &\vee \text{Complete}(\text{cown}) \end{aligned}$$

$$\text{Next} \triangleq \text{Terminating} \vee \exists c \in \text{Cowns} : \text{Run}(c) \vee \text{Unmute}$$

$$\begin{aligned} \text{Spec} &\triangleq \\ &\wedge \text{Init} \\ &\wedge \Box[\text{Next}]_{\text{vars}} \\ &\wedge \forall c \in \text{Cowns} : \text{WF}_{\text{vars}}(\text{Run}(c)) \\ &\wedge \text{WF}_{\text{vars}}(\text{Unmute}) \end{aligned}$$

Invariants

$$\begin{aligned} \text{MessageLimit} &\triangleq \\ &\text{LET } \text{msgs} \triangleq \text{ReduceSet}(\text{LAMBDA } c, \text{sum} : \text{sum} + \text{Len}(\text{queue}[c]), \text{Cowns}, 0) \text{IN} \\ &\text{msgs} \leq (\text{BehaviourLimit} + \text{Max}(\text{Cowns})) \end{aligned}$$

$$\begin{aligned} \text{RunningIsScheduled} &\triangleq \\ &\forall c \in \text{Cowns} : \text{running}[c] \Rightarrow \text{scheduled}[c] \wedge (c = \text{Max}(\text{CurrentMessage}(c))) \end{aligned}$$

$$\text{CownNotMutedBySelf} \triangleq \forall c \in \text{Cowns} : c \notin \text{mute}[c]$$

$$\text{LowPriorityMuted} \triangleq \forall c \in \text{Cowns} : (\text{priority}[c] = -1) \Rightarrow \text{Muted}(c)$$

$$\begin{aligned} \text{WillScheduleCown} &\triangleq \exists c \in \text{Cowns} : \\ &\vee \text{scheduled}[c] \\ &\vee \\ &\wedge \text{priority}[c] = -1 \\ &\wedge \exists k \in \text{DOMAIN } \text{mute} : (c \in \text{mute}[k]) \wedge (\text{priority}[k] = 0) \end{aligned}$$

$$\begin{aligned} \text{Nonblocking} &\triangleq \\ &\forall c \in \text{Cowns} : \forall m \in \text{Range}(\text{queue}[c]) : \\ &\neg(\exists h \in \text{HighPriority}(m) : \exists l \in \text{LowPriority}(m) : (h < c) \wedge (l \leq c)) \end{aligned}$$

$RunningNotBlocked \triangleq$
 $\forall c \in Cowns : running[c] \Rightarrow (\forall k \in CurrentMessage(c) : blocker[k] = Null)$

$Acquired(c) \triangleq \exists k \in Cowns : (k > c) \wedge (c \in \text{UNION } Range(queue[k]))$
 $UnscheduledByMuteOrAcquire \triangleq$
 $\forall c \in Cowns : \neg((priority[c] = -1) \vee Acquired(c)) \equiv scheduled[c]$

$BehaviourAcquisition \triangleq$
 $\forall c \in Cowns : \forall k \in \text{UNION } Range(queue[c]) : (k < c) \Rightarrow \neg scheduled[k]$

$AcquiredBy(a, b) \triangleq (a < b) \wedge (a \in \text{UNION } Range(queue[b]))$
 $AcquiredOnce \triangleq$
 $\forall a \in Cowns : \forall b \in Cowns : \forall c \in Cowns :$
 $(AcquiredBy(a, b) \wedge AcquiredBy(a, c)) \Rightarrow (b = c)$

$SelfInCurrentMessage \triangleq$
 $\forall c \in Cowns : (Len(queue[c]) > 0) \Rightarrow (c \in CurrentMessage(c))$

$HighPriorityInQueue \triangleq$
 $\forall c \in Cowns : (priority[c] = 1) \Rightarrow$
 $\exists k \in Cowns : c \in \text{UNION } Range(queue[k])$

$Required(c) \triangleq \exists k \in Cowns : (k < c) \wedge (c \in \text{UNION } Range(queue[k]))$
 $SleepingIsNormalOrRequired \triangleq$
 $\forall c \in Cowns : Sleeping(c) \Rightarrow ((priority[c] = 0) \vee Required(c))$

$MuteSetsDisjoint \triangleq$
 $\forall c \in Cowns : \forall k \in Cowns :$
 $((mute[c] \cap mute[k]) \neq \{\}) \Rightarrow (c = k)$

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

$$\text{IN} \quad \exists c \in \text{Cowns} : \langle c, c \rangle \in TC(s)$$

$$\text{MutedBy}(a, b) \triangleq (a \in \text{mute}[b]) \wedge (\text{priority}[a] = -1)$$

$$\text{AcyclicTCMute} \triangleq \neg \text{CyclicTransitiveClosure}(\text{MutedBy})$$

Temporal Properties

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

$$\text{SomeCownWillBeScheduled} \triangleq \Box \Diamond (\exists c \in \text{Cowns} : \text{scheduled}[c])$$
