

EXTENDS *FiniteSets*, *Integers*, *Sequences*, *TLC*

$Null \triangleq 0$
 $Cowns \triangleq 1 \dots 4$
 $BehaviourLimit \triangleq 4$
 $OverloadThreshold \triangleq 2$
 $PriorityLevels \triangleq \{-1, 0, 1\}$

$Min(s) \triangleq \text{CHOOSE } x \in s : \forall y \in s \setminus \{x\} : y > x$
 $Max(s) \triangleq \text{CHOOSE } x \in s : \forall y \in s \setminus \{x\} : y < x$

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

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

$EmptyQueue(c) \triangleq Len(queue[c]) = 0$

$Sleeping(c) \triangleq scheduled[c] \wedge EmptyQueue(c)$

$Available(c) \triangleq scheduled[c] \wedge \neg EmptyQueue(c)$

$Overloaded(c) \triangleq Len(queue[c]) > OverloadThreshold$

$CurrentMessage(c) \triangleq \text{IF } EmptyQueue(c) \text{ THEN } \{\} \text{ ELSE } Head(queue[c])$

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

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

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

RECURSIVE $Blockers(-)$

$Blockers(c) \triangleq$
 $\quad \text{IF } blocker[c] = Null \text{ THEN } \{\} \text{ ELSE } \{blocker[c]\} \cup Blockers(blocker[c])$
 $Prioritizing(cs) \triangleq$
 $\quad \text{LET } unprioritized \triangleq \{c \in cs : priority[c] < 1\} \text{ IN}$
 $\quad unprioritized \cup \text{UNION } \{Blockers(c) : c \in unprioritized\}$

$ValidMutor(c) \triangleq$
 $\quad \vee (priority[c] = 1) \wedge Overloaded(c)$
 $\quad \vee (priority[c] = -1)$

$Init \triangleq$
 $\quad \wedge fuel = BehaviourLimit$
 $\quad \wedge queue = [c \in Cowns \mapsto \{\{c\}\}]$

$\wedge \text{scheduled} = [c \in \text{Cowns} \mapsto \text{TRUE}]$
 $\wedge \text{running} = [c \in \text{Cowns} \mapsto \text{FALSE}]$
 $\wedge \text{priority} = [c \in \text{Cowns} \mapsto 0]$
 $\wedge \text{blocker} = [c \in \text{Cowns} \mapsto \text{Null}]$
 $\wedge \text{mutor} = [c \in \text{Cowns} \mapsto \text{Null}]$
 $\wedge \text{mute} = [c \in \text{Cowns} \mapsto \{\}]$

$\text{Terminating} \triangleq$
 $\wedge \forall c \in \text{Cowns} : \text{EmptyQueue}(c)$
 $\wedge \text{UNCHANGED vars}$

$\text{Acquire}(cown) \triangleq$
 $\text{LET } msg \triangleq \text{CurrentMessage}(cown) \text{ IN}$
 $\wedge \text{Available}(cown)$
 $\wedge cown < \text{Max}(msg)$
 $\wedge \text{IF } \text{priority}[cown] = 1 \text{ THEN}$
 $\quad \text{LET } \text{prioritizing} \triangleq \text{Prioritizing}(\{\text{Min}(\{c \in msg : c > cown\})\}) \text{ IN}$
 $\quad \text{LET } \text{unmuting} \triangleq \text{LowPriority}(\text{prioritizing}) \text{ IN}$
 $\quad \wedge \text{priority}' = [c \in \text{prioritizing} \mapsto 1] @@ \text{priority}$
 $\quad \wedge \text{scheduled}' = (cown :> \text{FALSE}) @@ [c \in \text{unmuting} \mapsto \text{TRUE}] @@ \text{scheduled}$
 ELSE
 $\quad \wedge \text{scheduled}' = (cown :> \text{FALSE}) @@ \text{scheduled}$
 $\quad \wedge \text{UNCHANGED } \langle \text{priority}, \text{mute} \rangle$
 $\wedge \text{LET } \text{next} \triangleq \text{Min}(\{c \in msg : c > cown\}) \text{ IN}$
 $\quad \wedge \text{blocker}' = (cown :> \text{next}) @@ \text{blocker}$
 $\quad \wedge \text{LET } q \triangleq (cown :> \text{Tail}(\text{queue}[cown])) @@ \text{queue} \text{ IN}$
 $\quad \quad \text{queue}' = (\text{next} :> \text{Append}(\text{queue}[\text{next}], msg)) @@ q$
 $\wedge \text{UNCHANGED } \langle \text{fuel}, \text{running}, \text{mutor}, \text{mute} \rangle$

$\text{Prerun}(cown) \triangleq$
 $\text{LET } msg \triangleq \text{CurrentMessage}(cown) \text{ IN}$
 $\wedge \text{scheduled}[cown]$
 $\wedge \neg \text{running}[cown]$
 $\wedge \text{IF } msg = \{\} \text{ THEN FALSE ELSE } cown = \text{Max}(msg)$
 $\wedge \text{priority}' = (cown :> \text{IF } \text{RequiresPriority}(cown) \text{ THEN } 1 \text{ ELSE } 0) @@ \text{priority}$
 $\wedge \text{running}' = (cown :> \text{TRUE}) @@ \text{running}$
 $\wedge \text{blocker}' = [c \in msg \mapsto \text{Null}] @@ \text{blocker}$
 $\wedge \text{UNCHANGED } \langle \text{fuel}, \text{queue}, \text{scheduled}, \text{mutor}, \text{mute} \rangle$

$\text{Send}(cown) \triangleq$
 $\text{LET } \text{senders} \triangleq \text{CurrentMessage}(cown) \text{ IN}$
 $\wedge \text{running}[cown]$
 $\wedge \text{fuel} > 0$
 $\wedge \exists \text{receivers} \in \text{SUBSET } \text{Cowns} :$
 $\quad \wedge \text{Cardinality}(\text{receivers}) > 0$
 $\quad \wedge \text{queue}' =$

```

    (Min(receivers):> Append(queue[Min(receivers)], receivers)) @@ queue
  ∧ IF priority[Min(receivers)] = 1 THEN
    LET prioritizing  $\triangleq$  Prioritizing({Min(receivers)})IN
    LET unmuting  $\triangleq$  LowPriority(prioritizing)IN
    ∧ priority' = [c ∈ prioritizing ↦ 1] @@ priority
    ∧ scheduled' = [c ∈ unmuting ↦ TRUE] @@ scheduled
    ∧ LET mutors  $\triangleq$  {c ∈ receivers \ senders : ValidMutor(c)}IN
      IF
        ∧ mutors ≠ {}
        ∧ mutor[cown] = Null
        ∧ ∀ c ∈ senders : priority[c] = 0
        ∧ ∀ c ∈ senders : c ∉ receivers TODO: justify
      THEN
        ∧ mutor' = (cown:> Min(mutors)) @@ mutor
      ELSE
        ∧ UNCHANGED ⟨mutor⟩
    ELSE
      ∧ UNCHANGED ⟨scheduled, priority, mutor⟩
  ∧ fuel' = fuel - 1
  ∧ UNCHANGED ⟨running, blocker, mute⟩

Complete(cown)  $\triangleq$ 
  LET msg  $\triangleq$  CurrentMessage(cown)IN
  ∧ running[cown]
  ∧ IF mutor[cown] ≠ Null THEN
    LET muting  $\triangleq$  {c ∈ msg : priority[c] = 0}IN
    ∧ priority' = [c ∈ muting ↦ -1] @@ priority
    ∧ mute' = (mutor[cown]:> mute[mutor[cown]] ∪ muting) @@ mute
    ∧ scheduled' = [c ∈ msg ↦ c ∉ muting] @@ scheduled
  ELSE
    ∧ scheduled' = [c ∈ msg ↦ TRUE] @@ scheduled
    ∧ priority' =
      (cown:> IF Len(queue[cown]) = 1 THEN 0 ELSE priority[cown]) @@
      [c ∈ msg \ {cown} ↦ IF EmptyQueue(c) THEN 0 ELSE priority[c]] @@
      priority
    ∧ UNCHANGED ⟨mute⟩
  ∧ queue' = (cown:> Tail(queue[cown])) @@ queue
  ∧ running' = (cown:> FALSE) @@ running
  ∧ mutor' = (cown:> Null) @@ mutor
  ∧ UNCHANGED ⟨fuel, blocker⟩

Unmute  $\triangleq$ 
  LET invalid_keys  $\triangleq$  {c ∈ DOMAIN mute : priority[c] = 0}IN
  LET unmuting  $\triangleq$  UNION Range([k ∈ invalid_keys ↦ LowPriority(mute[k])])IN
  ∧ unmuting ≠ {}

```

$$\begin{aligned}
&\wedge \text{priority}' = [c \in \text{unmuting} \mapsto 0] @@ \text{priority} \\
&\wedge \text{mute}' = [c \in \text{invalid_keys} \mapsto \{\}] @@ \text{mute} \\
&\wedge \text{scheduled}' = [c \in \text{unmuting} \mapsto \text{TRUE}] @@ \text{scheduled} \\
&\wedge \text{UNCHANGED } \langle \text{fuel}, \text{queue}, \text{running}, \text{blocker}, \text{mutor} \rangle
\end{aligned}$$

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

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

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

Utility Functions

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

$$\begin{aligned}
\text{ReduceSet}(\text{op}(_, _), \text{set}, \text{acc}) &\triangleq \\
&\text{LET } f[s \in \text{SUBSET } \text{set}] \triangleq \\
&\quad \text{IF } s = \{\} \text{ THEN } \text{acc} \text{ ELSE LET } x \triangleq \text{Pick}(s) \text{ IN } \text{op}(x, f[s \setminus \{x\}]) \\
&\text{IN } f[\text{set}]
\end{aligned}$$

$$\begin{aligned}
\text{MutedBy}(a, b) &\triangleq (a \in \text{mute}[b]) \wedge (\text{priority}[a] = -1) \\
\text{Muted}(c) &\triangleq \exists k \in \text{Cowns} : \text{MutedBy}(c, k)
\end{aligned}$$

$$\begin{aligned}
\text{AcquiredBy}(a, b) &\triangleq (a < b) \wedge (a \in \text{UNION } \text{Range}(\text{queue}[b])) \\
\text{Acquired}(c) &\triangleq \exists k \in \text{Cowns} : \text{AcquiredBy}(c, k)
\end{aligned}$$

$$\text{Required}(c) \triangleq \exists k \in \text{Cowns} : (k < c) \wedge (c \in \text{UNION } \text{Range}(\text{queue}[k]))$$

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

$$\text{TC}(R) \triangleq$$

$$\begin{aligned}
&\text{LET} \\
&\quad S \triangleq \{r[1] : r \in R\} \cup \{r[2] : r \in R\} \\
&\quad \text{RECURSIVE } \text{TCR}(-) \\
&\quad \text{TCR}(T) \triangleq \\
&\quad \quad \text{IF } T = \{\} \text{ THEN } R \\
&\quad \quad \text{ELSE} \\
&\quad \quad \quad \text{LET} \\
&\quad \quad \quad \quad r \triangleq \text{CHOOSE } s \in T : \text{TRUE} \\
&\quad \quad \quad \quad RR \triangleq \text{TCR}(T \setminus \{r\}) \\
&\quad \quad \text{IN}
\end{aligned}$$

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

IN

$$TCR(S)$$

$$\begin{aligned} &CyclicTransitiveClosure(R(-, -)) \triangleq \\ &\text{LET } s \triangleq \{\langle a, b \rangle \in Cowns \times Cowns : R(a, b)\} \\ &\text{IN } \exists c \in Cowns : \langle c, c \rangle \in TC(s) \end{aligned}$$

Temporal Properties

The model does not livelock.

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

Invariants

The message limit for *TLC* is enforced (the model has finite state space).

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

The running *cown* is scheduled and the greatest *cown* in the head of its queue.

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

A *cown* is not its own *mutor*.

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

A low-priority *cown* is muted.

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

A low-priority *cown* is not acquired.

$$LowPriorityNotAcquired \triangleq \forall c \in LowPriority(Cowns) : \neg Acquired(c)$$

There cannot be message that has acquired a high-priority *cown* and has acquired, or is in the queue of, a low-priority *cown*.

$$\begin{aligned} &Nonblocking \triangleq \\ &\forall c \in Cowns : \forall m \in Range(queue[c]) : \\ &\quad \forall \langle l, h \rangle \in LowPriority(m) \times HighPriority(m) : (c \leq h) \vee (c < l) \end{aligned}$$

All cowns in a running message have no blocker.

$$\begin{aligned} &RunningNotBlocked \triangleq \\ &\forall c \in Cowns : running[c] \Rightarrow (\forall k \in CurrentMessage(c) : blocker[k] = Null) \end{aligned}$$

An unscheduled *cown* is either muted or acquired.

$$\begin{aligned} &UnscheduledByMuteOrAcquire \triangleq \\ &\forall c \in Cowns : \neg((priority[c] = -1) \vee Acquired(c)) \equiv scheduled[c] \end{aligned}$$

A *cown* can only be acquired by at most one *cown*.

$$AcquiredOnce \triangleq$$

$$\forall \langle a, b, c \rangle \in \text{Cowns} \times \text{Cowns} \times \text{Cowns} : \\ (\text{AcquiredBy}(a, b) \wedge \text{AcquiredBy}(a, c)) \Rightarrow (b = c)$$

All messages in a *cown*'s queue must contain the *cown*.

$$\text{SelfInQueueMessages} \triangleq \forall c \in \text{Cowns} : \forall m \in \text{Range}(\text{queue}[c]) : c \in m$$

A *cown* is acquired by one of its blockers.

$$\text{Blocker} \triangleq \forall c \in \text{Cowns} : \\ \text{blocker}[c] \in \text{Cowns} \Rightarrow \exists k \in \text{Blockers}(c) : \text{AcquiredBy}(c, k)$$

All blockers of a high-priority *cown* are high-priority.

$$\text{HighPriorityBlockersAreHighPriority} \triangleq \\ \forall c \in \text{HighPriority}(\text{Cowns}) : \forall k \in \text{Blockers}(c) : \text{priority}[k] = 1$$

Warning: not enforced by implementation.

$$\text{SleepingIsNormal} \triangleq \forall c \in \text{Cowns} : \text{Sleeping}(c) \Rightarrow (\text{priority}[c] = 0)$$

High-priority cowns has messages in its queue or is acquired.

$$\text{HighPriorityHasWork} \triangleq \forall c \in \text{HighPriority}(\text{Cowns}) : \\ \vee \neg \text{EmptyQueue}(c) \\ \vee \text{Acquired}(c)$$

A muted *cown* has only one *mutor* in the mute map.

$$\text{MuteSetsDisjoint} \triangleq \forall \langle a, b \rangle \in \text{Cowns} \times \text{Cowns} : \\ ((\text{mute}[a] \cap \text{mute}[b]) \neq \{\}) \Rightarrow (a = b)$$

The transitive closure of the relation *MutedBy* has no cycles.

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