MODULE *backpressure*

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

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

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

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

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

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

$Sleeping(c) \triangleq (c \in scheduled) \land EmptyQueue(c)$

$Available(c) \triangleq (c \in scheduled) \land \neg EmptyQueue(c)$

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

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

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

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

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

RECURSIVE $Blockers(\_)$
$Blockers(c) \triangleq \text{IF } blocker[c] = Null \text{ THEN } \{\} \text{ ELSE } \{blocker[c]\} \cup Blockers(blocker[c])$
$Prioritise(c1) \triangleq \text{IF } priority[c1] < 1 \text{ THEN } \{c1\} \cup Blockers(c1) \text{ ELSE } \{\}$

$Mutor(c) \triangleq ((priority[c] = 1) \land Overloaded(c)) \lor (priority[c] = -1)$

$Init \triangleq$
$\quad \land\, fuel = BehaviourLimit$
$\quad \land\, queue = [c \in Cowns \mapsto \langle\{c\}\rangle]$
$\quad \land\, scheduled = \{c \in Cowns : \text{TRUE}\}$
$\quad \land\, running = \{\}$
$\quad \land\, priority = [c \in Cowns \mapsto 0]$
$\quad \land\, blocker = [c \in Cowns \mapsto Null]$
$\quad \land\, mutor = [c \in Cowns \mapsto Null]$
$\quad \land\, mute = [c \in Cowns \mapsto \{\}]$

$$
\begin{array}{l}
AcquireHigh(cown) \triangleq \\
\quad cown \in scheduled \\
\quad \land Len(queue[cown]) \neq 0 \\
\quad \land \text{LET } msg \triangleq Head(queue[cown]) \text{IN} \\
\quad \land cown < Max(msg) \\
\quad \land \text{LET } next \triangleq Min(\{c \in msg : c > cown\}) \text{IN} \\
\qquad \land priority[cown] = 1 \\
\qquad \land \text{LET } prioritizing \triangleq Prioritise(Min(\{c \in msg : c > cown\})) \text{IN} \\
\qquad\quad \text{LET } unmuting \triangleq \{c \in prioritizing : priority[c] = -1\} \text{IN} \\
\qquad\qquad \land\ priority' = [c \in prioritizing \mapsto 1] \mathbin{@@} priority \\
\qquad\qquad \land\ scheduled' = (scheduled \cup unmuting) \setminus \{cown\} \\
\qquad\qquad \land\ blocker' = (cown :> next) \mathbin{@@} blocker \\
\qquad\qquad \land\ queue' = (next :> Append(queue[next],\ msg)) \mathbin{@@} (cown :> Tail(queue[cown])) \mathbin{@@} queue \\
\qquad\qquad \land\ \text{UNCHANGED } \langle fuel,\ running,\ mutor,\ mute \rangle
\end{array}
$$

$$
\begin{array}{l}
AcquireNormal(cown) \triangleq \\
\quad cown \in scheduled \\
\quad \land Len(queue[cown]) \neq 0 \\
\quad \land \text{LET } msg \triangleq Head(queue[cown]) \text{IN} \\
\quad \land cown < Max(msg) \\
\quad \land \text{LET } next \triangleq Min(\{c \in msg : c > cown\}) \text{IN} \\
\qquad \land priority[cown] \neq 1 \\
\qquad \land scheduled' = scheduled \setminus \{cown\} \\
\qquad \land blocker' = (cown :> next) \mathbin{@@} blocker \\
\qquad \land queue' = (next :> Append(queue[next],\ msg)) \mathbin{@@} (cown :> Tail(queue[cown])) \mathbin{@@} queue \\
\qquad \land \text{UNCHANGED } \langle fuel,\ running,\ priority,\ mutor,\ mute \rangle
\end{array}
$$

$$
Acquire(cown) \triangleq AcquireHigh(cown) \lor AcquireNormal(cown)
$$

$$
\begin{array}{l}
StartHigh(cown) \triangleq \\
\quad cown \in scheduled \\
\quad \land cown \notin running \\
\quad \land Len(queue[cown]) \neq 0 \\
\quad \land RequiresPriority(cown) \\
\quad \land \text{LET } msg \triangleq Head(queue[cown]) \text{IN} \\
\qquad \land cown = Max(msg) \\
\qquad \land priority' = (cown :> 1) \mathbin{@@} priority \\
\qquad \land running' = running \cup \{cown\} \\
\qquad \land blocker' = [c \in msg \mapsto Null] \mathbin{@@} blocker \\
\qquad \land \text{UNCHANGED } \langle fuel,\ queue,\ scheduled,\ mutor,\ mute \rangle
\end{array}
$$

$$
\begin{array}{l}
StartNormal(cown) \triangleq \\
\quad cown \in scheduled \\
\quad \land cown \notin running \\
\quad \land Len(queue[cown]) \neq 0 \\
\quad \land \neg RequiresPriority(cown)
\end{array}
$$

$\land$ LET $msg \triangleq Head(queue[cown])$IN
$\land\ cown = Max(msg)$
$\land\ priority' = (cown :> 0) @@ priority$
$\land\ running' = running \cup \{cown\}$
$\land\ blocker' = [c \in msg \mapsto Null] @@ blocker$
$\land$ UNCHANGED $\langle fuel, queue, scheduled, mutor, mute\rangle$

$Start(cown) \triangleq StartHigh(cown) \lor StartNormal(cown)$

$SendAndMute(cown) \triangleq$ here senders can be empty
LET $senders \triangleq CurrentMessage(cown)$IN
$\land\ cown \in running$
$\land\ fuel > 0$
$\land \exists\, receivers \in$ SUBSET $Cowns : receivers \neq \{\}$
$\land$ LET $first \triangleq Min(receivers)$IN
$priority[first] = 1$
$\land\ mutor[cown] = Null$
$\land\ (\forall\, c \in senders : priority[c] = 0)$
$\land\ receivers \setminus senders = receivers$
$\land \exists\, c1 \in receivers : Mutor(c1) \land (\forall\, c2 \in receivers : c2 < c1 \Rightarrow \neg Mutor(c2))$
$\land\ mutor' = (cown :> c1) @@ mutor$
$\land$ LET $prioritizing \triangleq Prioritise(first)$IN
$scheduled' = scheduled \cup LowPriority(prioritizing)$
$\land\ priority' = [c \in prioritizing \mapsto 1] @@ priority$
$\land\ queue' = (first :> Append(queue[first], receivers)) @@ queue$
$\land\ fuel' = fuel - 1$
$\land$ UNCHANGED $\langle running, blocker, mute\rangle$

$SendNoMute(cown) \triangleq$ here senders can be empty
LET $senders \triangleq CurrentMessage(cown)$IN
$\land\ cown \in running$
$\land\ fuel > 0$
$\land \exists\, receivers \in$ SUBSET $Cowns : receivers \neq \{\}$
$\land$ LET $first \triangleq Min(receivers)$IN
$\land\ priority[first] = 1$
$\land\ (mutor[cown] \neq Null$
$\lor\ (\exists\, c \in senders : priority[c] \neq 0 \lor c \in receivers)$ *TODO*: justify
$\lor\ (\forall\, c \in receivers : \neg Mutor(c)))$
$\land$ LET $prioritizing \triangleq Prioritise(first)$IN
$scheduled' = scheduled \cup LowPriority(prioritizing)$
$\land\ priority' = [c \in prioritizing \mapsto 1] @@ priority$
$\land\ queue' = (first :> Append(queue[first], receivers)) @@ queue$
$\land\ fuel' = fuel - 1$
$\land$ UNCHANGED $\langle running, blocker, mute, mutor\rangle$

$SendNormal(cown) \triangleq$ here senders can be empty

$\quad$ LET $senders \triangleq CurrentMessage(cown)$IN
$\quad \land\ cown \in running$
$\quad \land\ fuel > 0$
$\quad \land\ \exists\, receivers \in$ SUBSET $Couns : receivers \neq \{\}$
$\quad\quad \land$ LET $first \triangleq Min(receivers)$IN
$\quad\quad\ priority[first] \neq 1$
$\quad\quad\quad \land\ queue' = (first :> Append(queue[first], receivers))\ @@\ queue$
$\quad \land\ fuel' = fuel - 1$
$\quad \land$ UNCHANGED $\langle scheduled, priority, mutor, running, blocker, mute\rangle$

$Send(cown) \triangleq SendAndMute(cown) \lor SendNoMute(cown) \lor SendNormal(cown)$

$CompleteMute(cown) \triangleq$
$\quad cown \in running$
$\quad \land\ mutor[cown] \neq Null$
$\quad \land$ LET $msg \triangleq CurrentMessage(cown)$IN
$\quad\quad$ LET $muting \triangleq \{c \in msg : priority[c] = 0\}$IN
$\quad\quad\quad \land\ \ priority' = [c \in muting \mapsto -1]\ @@\ priority$
$\quad\quad\quad \land\ \ mute' = (mutor[cown] :> mute[mutor[cown]] \cup muting)\ @@\ mute$
$\quad\quad\quad \land\ \ scheduled' = (scheduled \cup msg) \setminus muting$
$\quad \land\ queue' = (cown :> Tail(queue[cown]))\ @@\ queue$
$\quad \land\ running' = running \setminus \{cown\}$
$\quad \land\ mutor' = (cown :> Null)\ @@\ mutor$
$\quad \land$ UNCHANGED $\langle fuel, blocker\rangle$

$CompleteNormal(cown) \triangleq$
$\quad cown \in running$
$\quad \land\ mutor[cown] = Null$
$\quad \land$ LET $msg \triangleq CurrentMessage(cown)$IN
$\quad\quad \land\ scheduled' = scheduled \cup msg$
$\quad\quad \land\ priority' = (cown :>$ IF $Len(queue[cown]) = 1$ THEN 0 ELSE $priority[cown])\ @@$
$\quad\quad\quad\quad\quad\quad [c \in msg \setminus \{cown\} \mapsto$ IF $EmptyQueue(c)$ THEN 0 ELSE $priority[c]]\ @@$
$\quad\quad\quad\quad\quad\quad priority$
$\quad \land\ queue' = (cown :> Tail(queue[cown]))\ @@\ queue$
$\quad \land\ running' = running \setminus \{cown\}$
$\quad \land\ mutor' = (cown :> Null)\ @@\ mutor$
$\quad \land$ UNCHANGED $\langle fuel, blocker, mute\rangle$

$Complete(cown) \triangleq CompleteMute(cown) \lor CompleteNormal(cown)$

$Unmute \triangleq$
$\quad$ LET $invalid\_keys \triangleq \{c \in$ DOMAIN $mute : priority[c] = 0\}$IN
$\quad$ LET $unmuting \triangleq$ UNION $Range([k \in invalid_keys \mapsto LowPriority(mute[k])])$IN
$\quad \land\ unmuting \neq \{\}$
$\quad \land\ priority' = [c \in unmuting \mapsto 0]\ @@\ priority$
$\quad \land\ mute' = [c \in invalid\_keys \mapsto \{\}]\ @@\ mute$

$\land\ scheduled' = scheduled \cup unmuting$
$\land$ UNCHANGED $\langle fuel,\ queue,\ running,\ blocker,\ mutor \rangle$

$Run(cown) \triangleq$
$\lor\ Acquire(cown)$
$\lor\ Start(cown)$
$\lor\ Send(cown)$
$\lor\ Complete(cown)$

$Terminating \triangleq$
$TODO$: only require empty queue
$\land \forall\, c \in Cowns$: $EmptyQueue(c)$
$\land\ Assert(\forall\, c \in Cowns : Sleeping(c),$ "$Termination\ with\ unscheduled\ cowns$")
$\land \forall\, c \in Cowns : Sleeping(c)$
$\land$ UNCHANGED $vars$

$Next \triangleq \exists\, c \in Cowns : Run(c) \lor Unmute$

$Spec \triangleq$
$\land\ Init$
$\land\ \Box[Next \lor Terminating]_{vars}$
$\land\ \forall\, c \in Cowns : \mathrm{WF}_{vars}(Run(c))$
$\land\ \mathrm{WF}_{vars}(Unmute)$

Utility Functions

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

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

$MutedBy(a,\ b) \triangleq (a \in mute[b]) \land (priority[a] = -1)$
$Muted(c) \triangleq \exists\, k \in Cowns : MutedBy(c,\ k)$

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

$Required(c) \triangleq \exists\, k \in Cowns : (k < c) \land (c \in$ UNION $Range(queue[k]))$

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 ≜ CHOOSE s ∈ T : TRUE
           RR ≜ TCR(T \ {r})
         IN
           RR ∪ {⟨s, t⟩ ∈ S × S : ⟨s, r⟩ ∈ RR ∧ ⟨r, t⟩ ∈ RR}
    IN
      TCR(S)

CylcicTransitiveClosure(R(_, _)) ≜
  LET s ≜ {⟨a, b⟩ ∈ Cowns × Cowns : R(a, b)}
  IN  ∃ c ∈ Cowns : ⟨c, c⟩ ∈ TC(s)
```

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).

```
MessageLimit ≜
  LET msgs ≜ ReduceSet(LAMBDA c, sum : sum + Len(queue[c]), Cowns, 0)IN
  msgs ≤ (BehaviourLimit + Max(Cowns))
```

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

$RunningIsScheduled \triangleq$
$\quad \forall\, c \in running : (c \in scheduled) \wedge (c = Max(CurrentMessage(c)))$

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

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

A low-priority *cown* is muted.

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

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

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

All cowns in a running message have no blocker.

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

An unscheduled *cown* is either muted or acquired.

$UnscheduledByMuteOrAcquire \triangleq$
$\quad \forall\, c \in Cowns : (c \notin scheduled) \equiv ((priority[c] = -1) \vee Acquired(c))$

A *cown* in the queue of a greater *cown* is unscheduled.

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

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

$$AcquiredOnce \triangleq \forall\, \langle a,\, b,\, c\rangle \in Cowns \times Cowns \times Cowns : (AcquiredBy(a,\, b) \land AcquiredBy(a,\, c)) \Rightarrow (b = c)$$

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

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

A high-priority *cown* is in a queue of a high-priority *cown*.

$$HighPriorityInUnblockedQueue \triangleq \forall\, c \in HighPriority(Cowns) : \exists\, k \in HighPriority(Cowns) : c \in \text{UNION}\ Range(queue[k])$$

Warning: not enforced by implementation.

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

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

$$HighPriorityHasWork \triangleq \forall\, c \in HighPriority(Cowns) : \lor \neg EmptyQueue(c) \lor Acquired(c)$$

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

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

$$MutedByCycle(c1) \triangleq \text{LET}\ s \triangleq \{\langle a,\, b\rangle \in Cowns \times Cowns : MutedBy(a,\, b)\}\ \text{IN}\ \exists\, c2 \in Cowns : \langle c1,\, c2\rangle \in TC(s) \land \langle c2,\, c2\rangle \in TC(s)$$

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

$$MutedByIsAcyclic \triangleq \forall\, c \in Cowns : \neg MutedByCycle(c)$$

$$BlockerIsNextRequiredToRun \triangleq \forall\, c1,\, c2 \in Cowns : blocker[c1] = c2 \equiv \exists\, c3 \in Cowns : c3 > c1 \land \exists\, m \in Range(queue[c3]) : \neg(c3 \in running \land m = Head(queue[c3])) \land c1 \in m \land c2 = Min(\{c4 \in m : c4$$