MODULE *Validity*

EXTENDS *Naturals*, *FiniteSets*, *Commons*

CONSTANT *NPROCESSES*
CONSTANT *NGROUPS*
CONSTANT *NMESSAGES*
CONSTANT $CONFLICTR(\_, \_)$

This algorithm works in an environment with crash-stop failures, but we do not model processes failing. The set of all processes contains all correct ones.

LOCAL $Processes \triangleq 1 \mathrel{..} NPROCESSES$
LOCAL $Groups \triangleq 1 \mathrel{..} NGROUPS$
LOCAL $ProcessesInGroup \triangleq [g \in Groups \mapsto Processes]$

LOCAL $AllMessages \triangleq CreateMessages(NMESSAGES, Groups, Processes)$
LOCAL $MessagesCombinations \triangleq CreatePossibleMessages(AllMessages)$

VARIABLES *K*, *PreviousMsgs*, *Delivered*, *Votes*, *MemoryBuffer*,
QuasiReliableChannel, *AtomicBroadcastBuffer*

Initialize the instance for the Generic Multicast 2. The *INITIAL_MESSAGES* is a sequence, partially ordered. The sequence elements are sets of messages, messages that commute can share a set.

$Algorithm \triangleq$ INSTANCE *GenericMulticast2* WITH
$INITIAL\_MESSAGES \leftarrow [g \in Groups \mapsto$
$PartiallyOrdered($
$MessagesCombinations[(g \% NMESSAGES) + 1], CONFLICTR)]$

Weak fairness is necessary.

$Spec \triangleq Algorithm!SpecFair$

If a correct process GM-Cast a message m to $m.d$, then some process in $m.d$ eventually GM-Deliver m .

We verify that all messages on the messages that will be sent, then we verify that exists a process on the existent processes that did sent the message and eventually exists a process on $m.d$ *that delivers the message*.

$$
\begin{array}{l}
Validity \triangleq \\
\quad \forall m \in AllMessages : \\
\quad\quad m.o[1] \in Groups \land m.o[2] \in Processes \\
\quad\quad\quad \leadsto \exists g \in m.d : \\
\quad\quad\quad\quad \exists p \in ProcessesInGroup[g] : Algorithm!WasDelivered(g, p, m)
\end{array}
$$