MODULE *QuasiReliable*

This module is the abstraction for a quasi-reliable channel, the primary form of communication. Communication channels connect every pair of processes and provide two basic primitives to send and receive messages. The primitives *Send* and *Receive* have the following properties:

* No creation: for p_i , p_j , if p_j invokes Received m from p_i , then p_i must have invoked *Send* m to p_j ;
* No duplication: for p_i , p_j , for all *Send* m to p_j invoked by p_i , p_j invokes a corresponding Received from p_i is at most once;
* No loss: for p_i , p_j , if process p_i invokes *Send* m to p_j , and if neither p_i nor p_j fails, then eventually Received m from p_i is invoked in p_j .

LOCAL INSTANCE *Naturals*
LOCAL INSTANCE *Sequences*

Number of groups.
CONSTANT *NGROUPS*

Number of processes.
CONSTANT *NPROCESSES*

The set of initial messages.
CONSTANT *INITIAL_MESSAGES*

Represents the underlying network channel.
VARIABLE *QuasiReliableChannel*

A wrapper around the *Send* primitive. This procedure sends a message m to all processes in all groups. We do this instead of a single process to process to clear things up on the client side since all usages are to send messages to all participants.

$$
\begin{array}{l}
Send(m) \triangleq \\
\quad \land\ QuasiReliableChannel' = [\\
\quad\quad g \in \text{DOMAIN } QuasiReliableChannel \mapsto [\\
\quad\quad\quad p \in \text{DOMAIN } QuasiReliableChannel[g] \mapsto \\
\quad\quad\quad\quad QuasiReliableChannel[g][p] \cup \{m\}]]
\end{array}
$$

The receive primitive, using only this procedure, does not consume the message. We execute the callback passing the message existent in the specific process of the given group.

$$
\begin{array}{l}
Receive(g,\ p,\ Fn(\_)) \ \triangleq \\
\quad \land \ \exists\, m \in QuasiReliableChannel[g][p] : Fn(m)
\end{array}
$$

Bellow are some helper procedures built upon the *Send* and *Receive* primitives.

A wrapper to send the messages while applying a map function to the process' network buffer. We need this because we can not execute multiple operations to a variable in a single step. For example, removing and adding a message must be a single operation. In cases where we must consume and send a message in the network, we use this wrapper.

$$
\begin{array}{l}
SendMap(Fn(\_,\ \_)) \ \triangleq \\
\quad \land \quad QuasiReliableChannel' = [\\
\qquad g \in \text{DOMAIN}\ QuasiReliableChannel \mapsto [\\
\qquad\quad p \in \text{DOMAIN}\ QuasiReliableChannel[g] \mapsto \\
\qquad\qquad Fn(p,\ QuasiReliableChannel[g][p])]]
\end{array}
$$

This procedure causes the process in the given to consume the specific message.

$$
\begin{array}{l}
Consume(g,\ p,\ m) \ \triangleq \\
\quad \land \quad QuasiReliableChannel' = [\\
\qquad QuasiReliableChannel\ \text{EXCEPT}\ ![g][p] = @ \setminus \{m\}]
\end{array}
$$

This procedure put both the *Receive* primitive with the consume procedure together. For a received message, execute the callback and removes it from the buffer.

$$
\begin{array}{l}
ReceiveAndConsume(g,\ p,\ Fn(\_)) \ \triangleq \\
\quad \land\, Receive(g,\ p,\ \text{LAMBDA}\ m : Fn(m) \land Consume(g,\ p,\ m))
\end{array}
$$

Initialize the algorithm with all processes in all groups with the same set of messages.

$$
\begin{array}{l}
Init \ \triangleq \\
\quad \land\, QuasiReliableChannel = [\\
\qquad g \in 1\,..\,NGROUPS \mapsto [\\
\qquad\quad p \in 1\,..\,NPROCESSES \mapsto INITIAL\_MESSAGES]]
\end{array}
$$