MODULE *GenericBroadcast*

This module is the abstraction for the Generic Broadcast, a primitive for group communication. A process can broadcast a message to a single group, and using conflict relation processes may order the delivery order.

We use a combination of sequences; each position contains a set; each set contains commuting messages. The former has an order, whereas the latter is unordered. With this approach, we have a generic delivery.

LOCAL INSTANCE *Naturals*
LOCAL INSTANCE *Sequences*
LOCAL INSTANCE *FiniteSets*
LOCAL INSTANCE *Commons*

CONSTANT *NGROUPS*
CONSTANT *NPROCESSES*
CONSTANT *INITIAL_MESSAGES*

The conflict relation to identify commuting messages.
CONSTANT *CONFLICTR*(_, _)

The underlying buffer that holds all the messages.
VARIABLE *GenericBroadcastBuffer*

We consume the message in the given group. If the set in the head is empty, we remove it; we remove only m otherwise.

LOCAL $Consume(S,\ m) \triangleq$
    IF $Cardinality(Head(S)) > 1$ THEN $ReplaceAt(S,\ 1,\ Head(S) \setminus \{m\})$
     ELSE $SubSeq(S,\ 2,\ Len(S))$

Verify if exists conflict in the process for the message.

LOCAL $ConflictIn(V,\ m) \triangleq \exists\, \langle n,\ x,\ y \rangle \in V : CONFLICTR(m,\ n)$
LOCAL $HasConflict(S,\ m) \triangleq$
    $Len(SelectSeq(S,\ \text{LAMBDA}\ V : ConflictIn(V,\ m[1]))) \neq 0$

We insert a message to the specific process' buffer. If the buffer is empty or there is a conflict, we add the message to the back of the sequence; otherwise, we add the message in the head.

```
LOCAL Insert(S, m) ≜
    IF Len(S) = 0 ∨ HasConflict(S, m) THEN Append(S, {m})
     ELSE ReplaceAt(S, Len(S), S[Len(S)] ∪ {m})
```

Broadcast a message to the given group. We insert the message in the buffer of all processes within this group.

```
GBroadcast(g, m) ≜
    ∧ GenericBroadcastBuffer' = [
        GenericBroadcastBuffer EXCEPT ![g] = [
            i ∈ 1 .. Len(GenericBroadcastBuffer[g]) ↦
                Insert(GenericBroadcastBuffer[g][i], m)]]
```

Generic deliver primitive to the process in the specific group. If the buffer is not empty, we invoke the call with the appropriate message and then consume it.

```
GBDeliver(g, p, Fn(_)) ≜
    ∧ Len(GenericBroadcastBuffer[g][p]) > 0
    ∧ Cardinality(Head(GenericBroadcastBuffer[g][p])) > 0
    ∧ LET
        Since messages in the same set commute, we can choose any.
        m ≜ CHOOSE v ∈ Head(GenericBroadcastBuffer[g][p]) : TRUE
      IN
        ∧ Fn(m)
        ∧ GenericBroadcastBuffer' = [
            GenericBroadcastBuffer EXCEPT ![g][p] =
                Consume(GenericBroadcastBuffer[g][p], m)]
```

Initialize the algorithm with the configuration values. The processes within a group will have the same sequence of messages.

```
Init ≜
    ∧ GenericBroadcastBuffer = [
        g ∈ 1 .. NGROUPS ↦ [
            p ∈ 1 .. NPROCESSES ↦ INITIAL_MESSAGES[g]]]
```